

TERCERA | EDICIÓN

# ADMINISTRACIÓN DE BASES DE DATOS DISEÑO Y DESARROLLO DE APLICACIONES



Michael V. Mannino



# Administración de bases de datos

Diseño y desarrollo de aplicaciones



# Administración de bases de datos

Diseño y desarrollo de aplicaciones

Tercera edición

**Michael V. Mannino**  
*University of Colorado at Denver*

Revisión técnica  
**Carlos Villegas Quezada**  
*Universidad Iberoamericana, Ciudad de México*



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

**Director Higher Education:** Miguel Ángel Toledo Castellanos

**Director editorial:** Ricardo del Bosque Alayón

**Editor sponsor:** Pablo E. Roig Vázquez

**Editora de desarrollo:** Ana Laura Delgado Rodríguez

**Supervisor de producción:** Zeferino García García

**Traducción:** Ekaterina Guerrero Ushakova

José Julián Díaz Díaz

## **ADMINISTRACIÓN DE BASES DE DATOS.**

**Diseño y desarrollo de aplicaciones**

**Tercera edición**

Prohibida la reproducción total o parcial de esta obra,  
por cualquier medio, sin la autorización escrita del editor.



DERECHOS RESERVADOS © 2007 respecto a la primera edición en español por  
McGRAW-HILL/INTERAMERICANA EDITORES, S.A. DE C.V.

*A Subsidiary of The McGraw-Hill Companies, Inc.*

Edificio Punta Santa Fe  
Prolongación Paseo de la Reforma 1015, Torre A  
Piso 17, Colonia Desarrollo Santa Fe,  
Delegación Álvaro Obregón  
C.P. 01376, México, D.F.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana, Reg. Núm. 736

**ISBN-13: 978-970-10-6109-1**

**ISBN-10: 970-10-6109-8**

Traducido de la tercera edición de *Database Design, Application Development, & Administration*  
Copyright © 2007 by The McGraw-Hill Companies, Inc. All rights reserved.

ISBN-13: 978-0-07-294220-0

ISBN-10: 0-07-294220-7

1234567890

09865432107

Impreso en México

*Printed in Mexico*

Deseo dedicar este libro a mis hijas Julia y Aimee. Sus sonrisas y afecto me inspiran día con día.

# Acerca del autor

---

Michael V. Mannino ha estado involucrado en el campo de las bases de datos desde 1980. Ha impartido cursos de administración de bases de datos desde 1983 en diversas universidades de renombre (University of Florida, University of Texas en Austin, University of Washington, University of Colorado en Denver). Su audiencia incluye a estudiantes de licenciatura y maestría en Administración de Sistemas de Información (MIS), maestría en Administración (MBA) y programas de doctorado, además de cursos corporativos en programas de reentrenamiento. Ha estado activo en la investigación de bases de datos, lo que se evidencia por sus publicaciones en importantes revistas del IEEE (*Transactions on Knowledge and Data Engineering* y *Transactions on Software Engineering*), ACM (*Communications y Computing Surveys*) e INFORMS (*Informs Journal on Computing* e *Information Systems Research*). Su investigación abarca varias encuestas populares y artículos de tutoría, así como múltiples documentos que describen trabajo original de investigación. En el capítulo 12 del presente libro han sido incorporados los resultados prácticos de su investigación sobre una metodología orientada a formas aplicadas al diseño de las bases de datos.

# Prefacio

---

## Ejemplo de motivación

---

Paul Hong, propietario de International Industrial Adhesives, Inc., se encuentra eufórico por el desempeño reciente de su negocio, pero toma precauciones respecto a sus perspectivas futuras. Sus ingresos y ganancias crecen más allá de cualquier pronóstico mientras que sus gastos se mantienen fijos. Él atribuye esta situación a la recuperación económica internacional, al uso de *outsourcing* para focalizar sus recursos y al desarrollo estratégico de las tecnologías de la información. Su euforia sobre el desempeño reciente está atemperado por las perspectivas futuras. El éxito de su negocio ha atraído a nuevos competidores que se dirigen hacia sus clientes más rentables. El resultado de las nuevas y costosas iniciativas de la industria en el comercio electrónico es incierto. Las nuevas regulaciones gubernamentales incrementarían significativamente el costo de su operación como empresa pública, lo que amenaza sus planes de una oferta pública inicial para aumentar su capital. A pesar de la euforia por el éxito reciente de su negocio, permanece cauteloso respecto a las nuevas direcciones que debe tomar con el fin de asegurar el crecimiento continuo de su empresa.

Paul Hong necesita evaluar las inversiones en tecnologías de la información para mantenerse adelante de sus competidores y controlar los costos de los mandatos industriales y gubernamentales. Para igualar a sus competidores, necesita datos más detallados y oportunos acerca de las tendencias de la industria, las acciones de los competidores y las operaciones de los distribuidores. Él desea encontrar una solución costo-efectiva para soportar la iniciativa de la industria por el comercio electrónico. Para preparar su operación como empresa pública, debe realizar auditorías de tecnologías de la información y cumplir con los requisitos de información que exige el gobierno a este tipo de empresas. Por todo esto, él no está seguro sobre el uso de estándares y tecnologías propietarias o no propietarias.

Estas preocupaciones implican el uso significativo de la tecnología de bases de datos como parte de la creciente infraestructura de cómputo en las empresas. Las características de procesamiento de transacciones en los DBMS empresariales proporcionan el fundamento para asegurar la confiabilidad del procesamiento en línea de los pedidos con el fin de apoyar las iniciativas empresariales para el cada vez más frecuente uso del comercio electrónico. Las características de los data warehouse en los DBMS empresariales proporcionan los fundamentos para los grandes data warehouse y para la captura de datos fuente en tiempo real. La tecnología de bases de datos paralelas puede mejorar el desempeño y la confiabilidad de las consultas de procesamiento de transacciones y de almacenes de datos por medio de adiciones incrementales en la capacidad de cálculo. Las características de las bases de datos de objetos proporcionan la habilidad para administrar grandes colecciones de documentos XML generados por las iniciativas de la industria enfocadas al comercio electrónico.

No obstante, las soluciones para las preocupaciones de Paul Hong no se encuentran únicamente en la tecnología. El uso del nivel apropiado de tecnología implica la visión para el futuro de la organización, una comprensión más profunda de la misma y las habilidades tradicionales de administración que permitan controlar el riesgo. Paul Hong se da cuenta de que su mayor reto es mezclar estas habilidades de manera tal que las soluciones eficaces puedan desarrollarse para International Industrial Adhesives, Inc.

## Introducción

---

Este libro de texto ofrece los fundamentos para comprender la tecnología de bases de datos que da apoyo a cuestiones empresariales de cómputo tales como las que enfrenta Paul Hong. Como

nuevo alumno de la administración de bases de datos, es necesario que usted comprenda primero los conceptos fundamentales de la administración de bases de datos y del modelo relacional de datos. Después, necesita dominar sus habilidades en el diseño de bases de datos y en el desarrollo de aplicaciones para bases de datos. Este libro de texto le proporciona las herramientas para ayudarle a comprender las bases de datos relacionales y adquirir las habilidades necesarias para resolver los problemas básicos y avanzados en la formulación de consultas, modelado de datos, normalización y requerimientos de los datos de las aplicaciones, además de la personalización de las aplicaciones de bases de datos.

Después de establecer dichas habilidades, usted estará listo para estudiar la función que tienen los especialistas de bases de datos y los ambientes de procesamiento en los que se utilizan las bases de datos. Este libro presenta las tecnologías fundamentales de las bases de datos en cada ambiente de procesamiento y relaciona estas tecnologías con los avances más recientes del comercio electrónico y el cómputo empresarial.

## Lo nuevo en la tercera edición

---

La tercera edición tiene revisiones significativas hechas a la segunda edición, y a la vez conserva la probada pedagogía de las otras dos ediciones. La experiencia obtenida al haber dado clases a alumnos de licenciatura y cursos superiores, así como la retroalimentación recibida por quienes adoptaron la segunda edición, han llevado al desarrollo de nuevo material y refinamientos del anterior. Los cambios más significativos hechos en la tercera edición se relacionan con los capítulos destinados al desarrollo de las bases de datos (capítulos 5 al 8): las reglas de negocios en el modelado de datos, lineamientos para analizar las necesidades de información de las empresas, cobertura expandida de los errores de diseño en el modelado de datos, cobertura expandida de la identificación de la dependencia funcional, y nueva cobertura en cuanto a las recomendaciones para optimizar las consultas. Esta nueva cobertura fortalece la metodología probada en la segunda edición, misma que proporcionaba la separación de la estructura de los diagramas de entidad-relación de la práctica del modelado empresarial de datos, una herramienta de modelado de datos personalizado (ER Assistant) para eliminar la confusión de la notación y el énfasis en la normalización como herramienta de refinamiento para el desarrollo de las bases de datos.

Para el desarrollo de aplicaciones de bases de datos, la tercera edición ofrece SQL:2003, un cambio revolucionario con respecto a SQL:1999. La tercera edición explica el alcance de SQL:2003, la dificultad de conformidad con el estándar y los nuevos elementos del mismo. Numerosos refinamientos de la cobertura para el desarrollo de aplicaciones de bases de datos amplían la cobertura probada de las dos primeras ediciones: lineamientos para la formulación de consultas, problemas de enlace avanzados, recomendaciones para la formulación de consultas para informes y formularios jerárquicos, y disparadores para restricciones suaves.

Para la administración de bases de datos y los entornos de procesamiento, la tercera edición proporciona cobertura expandida para la nueva tecnología de SQL:2003 y de Oracle 10g. Los temas nuevos más significativos son la tecnología de bases de datos paralelas, la cobertura ampliada para la reescritura de consultas para vistas materializadas y la transparencia en las bases distribuidas de Oracle. Se proporciona cobertura revisada para el control de candados, la verificación de puntos de recuperación de una base de datos, el tiempo de interacción del usuario en el diseño de la transacción, la representación del tiempo en las tablas de dimensión, la madurez del almacén de datos, los servicios web en el procesamiento de bases de datos cliente-servidor y la aceptación comercial de las arquitecturas de bases de datos de objetos.

Además del nuevo material y de los refinamientos hechos al que ya existía, la tercera edición extiende los complementos de cada capítulo. Esta edición contiene preguntas y problemas nuevos al final de cada capítulo, además de los resúmenes de sintaxis de SQL:2003. El nuevo material del sitio web de este libro de texto incluye casos de estudio, tareas para el primero y segundo cursos de bases de datos, y exámenes muestra.

La tercera edición tiene una organización capitular más fina con siete partes, para poder ofrecer bloques de aprendizaje más pequeños. La parte 1 abarca material introductorio acerca de la administración de bases de datos y el desarrollo de las mismas con el fin de proporcionar una

base conceptual para el conocimiento y habilidades detalladas en los capítulos subsecuentes. La parte 2 incluye los elementos esenciales para el modelo de datos relacional para la creación de bases de datos y la formulación de consultas. El desarrollo de las bases de datos se divide entre el modelado de datos de la parte 3 y el diseño de tablas físicas de la parte 4. La parte 5 abarca un desarrollo más avanzado de aplicaciones que cubre los problemas de enlace, vistas de las bases de datos, procedimientos y disparadores almacenados. La parte 6 incluye información sobre el desarrollo de bases de datos con una integración de la vista y un estudio de caso bastante amplio. La parte 7 abarca la administración de la base de datos y los ambientes de procesamiento para los DBMSs, material que fue presentado en la parte 4 de la segunda edición.

## Ventajas competitivas

---

Este libro de texto ofrece características sobresalientes que no tienen par en los libros de la competencia. Las características únicas son una cobertura amplia de SQL tanto para Access como para Oracle, lineamientos para la solución de problemas para ayudar a la adquisición de habilidades fundamentales, bases de datos muestra y ejemplos diseñados con todo cuidado, un completo estudio de caso, cobertura de temas avanzados, material integrado de laboratorio y el ER Assistant. Estas características proveen al lector de un paquete completo para un curso introductorio de bases de datos. Cada una de estas características se describe con mayor detalle en la lista que se presenta a continuación, al tiempo que la tabla P.1 resume las ventajas competitivas por capítulo.

- **Cobertura de SQL.** La profundidad y amplitud de la cobertura de SQL en este texto no tiene igual en libros de la competencia. La tabla P.2 resume dicha cobertura por capítulo. Las partes 2 y 5 proporcionan una cobertura vasta de las sentencias CREATE TABLE, SELECT, UPDATE, INSERT, DELETE, CREATE VIEW y CREATE TRIGGER. Se presentan numerosos ejemplos de problemas básicos, intermedios y avanzados.
- **Cobertura de Access y Oracle.** Los capítulos de las partes 2 y 5 ofrecen una cobertura detallada de Access y Oracle SQL. Se muestran ejemplos para ambos sistemas de administración de bases de datos para las sentencias SELECT, INSERT, UPDATE, DELETE y CREATE VIEW. Hay una cobertura significativa de las nuevas características de Oracle 10g

**TABLA P.1 Resumen de ventajas competitivas por capítulo**

Capítulo	Características únicas
2	Capítulo único que proporciona una introducción conceptual al proceso de desarrollo de bases de datos
3	Representación visual de los operadores del álgebra relacional
4	Lineamientos para la formulación de bases de datos; cobertura para Oracle SQL, Access y SQL:2003
5	Énfasis en la notación ERD, reglas del negocio, reglas de diagramación con apoyo en el ER Assistant
6	Estrategias para analizar las necesidades de información del negocio, transformación del modelado de datos y detección de errores comunes de diseño
7	Lineamientos y procedimientos de normalización
8	Reglas de selección de índices, lineamientos de afinación de SQL, cobertura integrada de la optimización de consultas, estructuras de archivos y selección de índices
9	Lineamientos de formulación de consultas, cobertura para Oracle 10g, Access y SQL:2003; cobertura de temas avanzados de consultas anidadas, problemas de división y administración del valor nulo
10	Reglas para vistas actualizables, lineamientos para el requerimiento de datos para formularios e informes
11	Capítulo único que incluye conceptos y prácticas de los lenguajes de programación de bases de datos, procedimientos almacenados y disparadores
12	Capítulo único que abarca conceptos y prácticas relacionados con la integración y diseño de las vistas
13	Capítulo único que proporciona un caso de estudio completo acerca del procesamiento de préstamos estudiantiles
14	Lineamientos para los procesos importantes empleados por los administradores de bases de datos
15	Lineamientos para el diseño de transacciones y cobertura de temas avanzados
16	Modelo de madurez del data warehouse para evaluar el impacto de la tecnología en las organizaciones, cobertura de temas avanzados de características de bases de datos relacionales para el procesamiento y refrescado de data warehouses, cobertura extensa del data warehouse en Oracle 10g
17	Cobertura integrada del procesamiento de cliente-servidor, procesamiento paralelo de bases de datos, y bases de datos distribuidas
18	Cobertura de temas avanzados de características de objetos-relacionales en SQL:2003 y Oracle 10g

**TABLA P.2**  
**Cobertura de sentencias SQL por capítulo**

Capítulo	Sentencias SQL
3	CREATE TABLE
4	SELECT, INSERT, UPDATE, DELETE
9	SELECT (consultas anidadas, enlaces externos, administración de valores nulos); cobertura de Access, Oracle 10g y SQL:2003
10	CREATE VIEW, consultas y sentencias de manipulación que emplean vistas
11	CREATE PROCEDURE (Oracle), CREATE TRIGGER (Oracle y SQL:2003)
14	GRANT, REVOKE, CREATE ROLE, CREATE ASSERTION, cláusula CHECK de la sentencia CREATE TABLE, CREATE DOMAIN
15	COMMIT, ROLLBACK, SET TRANSACTION, SET CONSTRAINTS, SAVEPOINT
16	CREATE MATERIALIZED VIEW (Oracle), extensiones de la cláusula GROUP BY (Oracle y SQL:2003), CREATE DIMENSION (Oracle)
18	CREATE TYPE, CREATE TABLE (tablas capturadas y subtablas), SELECT (identificadores de objetos, expresiones de trayectoria, operador diferencial); cobertura SQL:2003 y Oracle 10g

**TABLA P.3**  
**Lineamientos para la solución de problemas por capítulo**

Capítulo	Lineamientos para la solución de problemas
3	Representación visual de las relaciones y de los operadores del álgebra relacional
4	Proceso de evaluación conceptual, preguntas de formulación de consultas
5	Reglas de diagramación
6	Lineamientos para analizar las necesidades de información de los negocios, transformaciones de diseño, identificación de errores comunes de diseño, reglas de conversión
7	Lineamientos para identificar las dependencias funcionales, procedimiento de síntesis simple
8	Reglas de selección de índices, lineamientos de afinación de SQL
9	Lineamientos de formulación de problemas de diferencia, evaluación de consultas anidadas, método de conteo para problemas de división
10	Reglas para consultas de enlace actualizables, pasos para analizar los requerimientos de datos en formularios e informes
11	Procedimiento de ejecución de disparadores
12	Pasos para el análisis de formularios, estrategias de integración de vistas
14	Lineamientos para administrar los procedimientos y disparadores almacenados; proceso de planeación de datos, proceso de selección de DBMS
15	Línea de tiempo de la transacción, lineamientos de diseño de transacción
16	Lineamientos para las representaciones de bases de datos relacionales de datos multidimensionales, lineamientos para la representación del tiempo en tablas de dimensión, desventajas para la actualización de un almacén de datos
17	Progresión de los niveles de transferencia para bases de datos distribuidas
18	Arquitecturas de bases de datos orientadas a objetos, comparación entre representación relacional y representación orientada a objetos

SQL que aparece en los capítulos 8, 9, 11, 15, 16 y 18. Además, los capítulos de las partes 2 y 5 cubren la sintaxis de SQL:2003 para dar apoyo a la instrucción que se dé con otros prominentes sistemas de administración de bases de datos.

- **Lineamientos de solución de problemas.** Los alumnos necesitan más que explicaciones de conceptos y ejemplos para resolver problemas. Ellos requieren de lineamientos que les ayuden a estructurar su proceso de pensamiento para que puedan atacar los problemas de manera sistemática. Los lineamientos proporcionan modelos mentales que ayudan a los alumnos a aplicar los conceptos a la solución de problemas básicos y avanzados. La tabla P.3 resume los lineamientos para la solución de problemas que aparecen en cada capítulo.
- **Bases de datos muestra y ejemplos.** A lo largo de los capítulos de las partes 2 y 5, se emplean dos bases de datos para proporcionar consistencia y continuidad. La base de datos de la universidad se emplea en los ejemplos a lo largo del capítulo, mientras que la base de datos de captura de órdenes se emplea en los problemas al final del capítulo. Los numerosos ejemplos y problemas con estas bases de datos resaltan las habilidades fundamentales en la formulación de consultas y los requerimientos de datos para las aplicaciones. Las versiones revisadas en las bases de datos proporcionan la separación entre los ejemplos básicos y los

avanzados. El sitio web contiene las sentencias CREATE TABLE, datos muestra, sentencias de manipulación de datos y archivos de bases de datos en Access para ambas bases de datos. Los capítulos de las partes 3, 4 y 7 utilizan bases de datos adicionales para ampliar la exposición del alumno a situaciones de negocios más diversas. Los alumnos necesitan exponerse a varias situaciones empresariales para adquirir las habilidades indispensables para el diseño de las bases de datos y comprender los conceptos más importantes para los especialistas en bases de datos. Otras bases de datos, que abarcan operaciones de servicios de agua, visitas a pacientes, revisiones a documentos académicos, rastreo financiero personal, reservaciones en líneas aéreas, oficinas de colocaciones, aseguradoras de automóviles, rastreo de ventas en almacenes y ventas de bienes raíces, complementan las bases de datos de la universidad y de captura de órdenes en los ejemplos de los capítulos y en los problemas al final del capítulo.

- **Estudio de caso completo.** El caso de Student Loan Limited se encuentra al final de la parte 6. La descripción del caso y su solución integran los conceptos que los alumnos aprendieron en los 12 capítulos anteriores sobre el desarrollo de aplicaciones y el diseño de bases de datos. Los problemas de seguimiento que aparecen al final del capítulo proporcionan oportunidades adicionales para que los estudiantes apliquen su conocimiento en un caso realista.
- **Laboratorio opcional integrado.** La administración de bases de datos se enseña mejor cuando los conceptos se ligan estrechamente con la práctica del diseño e implementación de bases de datos que emplea un DBMS. Para ayudar a los alumnos a aplicar los conceptos descritos en el libro de texto, se encuentran a su disposición un CD y el sitio web con material de laboratorio complementario. El CD contiene material de laboratorio para cuatro versiones de Microsoft Access (97, 2000, 2002 y 2003), así como bases de datos y ejercicios de práctica. Los materiales de laboratorio de Microsoft Access integran la cobertura detallada de Access con los conceptos de desarrollo de aplicaciones que describen en las partes 2 y 5.
- **Herramienta gratuita para el modelado de datos.** El ER Assistant ofrece una interfase simple para dibujar y analizar los diagramas de entidad-relación del modelado de datos, como se presentan en los capítulos de la parte 3. Con este programa, los alumnos pueden volverse productivos con gran rapidez, lo que les permite enfocarse en los conceptos del modelado de datos en lugar de los detalles de una herramienta CASE complicada. Para ayudar a los alumnos a que no cometan errores de diagramación, el ER Assistant confirma las reglas de diagramación presentadas en el capítulo 5.
- **Temas actuales e incisivos.** El libro cubre algunos temas que no aparecen en otros libros de texto de la competencia: formulación avanzada de consultas, vistas actualizables, desarrollo y administración de procedimientos y disparadores almacenados, requisitos de datos para los formularios e informes de captura de datos, integración de vistas, administración para el proceso de actualización para los data warehouses, modelo de madurez de un data warehouse, arquitecturas de bases de datos paralelas, arquitecturas de bases de datos orientadas a objetos, características de los data warehouses en SQL:2003 y Oracle 10g, y principios de diseño de transacciones. Estos temas permiten que los alumnos motivados obtengan una comprensión más amplia de la administración de las bases de datos.
- **Paquete completo para el curso.** Según sea el criterio del curso, algunos alumnos intentarán comprar hasta cinco libros para un curso introductorio de bases de datos: un libro de texto que abarque los principios, manuales de laboratorio que cubran los detalles de un DBMS y una herramienta CASE, un libro complementario de SQL y un libro de casos con problemas reales de práctica. Este libro de texto y su material complementario proporcionan una fuente más completa, integrada y menos costosa para el alumno.

## Audiencia del texto

---

Este libro ha sido diseñado para alumnos que cursan la licenciatura o cursos superiores de administración de bases de datos. En el nivel licenciatura, los alumnos deben tener una concentración (mayor o menor) o interés activo en los sistemas de información. Para las instituciones de dos

años, el instructor quizá quiera saltarse los temas avanzados y poner más énfasis en el manual opcional de laboratorio de Access. Los alumnos de licenciatura deberán tener un primer curso que cubra los conceptos fundamentales, hojas de cálculo, procesadores de palabras y posiblemente una breve introducción a las bases de datos. Un curso previo de programación puede resultar de utilidad (excepto para el capítulo 11), aunque no es obligatorio. Los demás capítulos hacen referencia a ciertos conceptos de programación, pero no se incluye el código de escritura. Para entender plenamente el capítulo 11, es esencial contar con experiencia previa en programación. Sin embargo, los conceptos básicos del capítulo 11 pueden abarcarse incluso si los alumnos no tienen conocimientos previos de programación de computadoras.

En niveles superiores, este libro es adecuado para los programas de maestría en administración o maestría en ciencias (en sistemas de información). El material avanzado de este libro debe ser especialmente adecuado para los alumnos que estudian la maestría en ciencias.

## Organización

---

Como el título lo indica, *Administración de base de datos. Diseño y desarrollo de aplicaciones* enfatiza las tres series de actividades. Antes de adquirirlas, los alumnos necesitan tener un fundamento de los conceptos básicos. La parte 1 proporciona los antecedentes conceptuales para un estudio detallado subsecuente del diseño de bases de datos, el desarrollo de aplicaciones para bases de datos y la administración de bases de datos. Los capítulos de la parte 1 presentan los principios de la administración de bases de datos y una visión general conceptual del proceso de desarrollo de bases de datos.

La parte 2 ofrece conocimiento funcional acerca del modelo relacional de datos. El capítulo 3 abarca la definición de tablas, las reglas de integridad y los operadores para recuperar la información útil de las bases de datos relacionales. El capítulo 4 presenta lineamientos para la formulación de consultas y numerosos ejemplos de sentencias en SQL.

Las partes 3 y 4 enfatizan las habilidades prácticas y los lineamientos de diseño para el proceso de desarrollo de bases de datos. Los estudiantes que deseen hacer una carrera como especialistas en bases de datos deberán tener la capacidad de desempeñar cada uno de los pasos del proceso de desarrollo de bases de datos. Deberán aprender las habilidades del modelado de datos, la conversión de esquemas, la normalización y el diseño físico de las bases de datos. Los capítulos de la parte 3 (capítulos 5 y 6) abarcan el modelado de datos por medio del modelo de entidad-relación. El capítulo 5 incluye la estructura de los diagramas de entidad-relación, mientras que el capítulo 6 presenta el uso de los diagramas de entidad-relación para analizar las necesidades de información del negocio. Los capítulos de la parte 4 (capítulos 7 y 8) presentan los principios y práctica del diseño de tablas para el diseño lógico y físico. El capítulo 7 estudia la motivación, las dependencias funcionales, los formatos normales y las consideraciones prácticas de la normalización de datos. El capítulo 8 ofrece una amplia cobertura del diseño físico de las bases de datos, incluidos objetivos, entradas, estructura de archivos y antecedentes para la optimización de consultas, así como importantes opciones de diseño.

La parte 5 proporciona un fundamento para la construcción de aplicaciones para bases de datos, de modo que ayuda a los alumnos a adquirir las habilidades necesarias para la formulación de consultas, la especificación de requerimientos de los formularios e informes de captura de datos, y los disparadores de codificación y procedimientos almacenados. En el capítulo 9 se observan ejemplos adicionales de SQL intermedio y avanzado, junto con las habilidades de formulación de consultas correspondientes. El capítulo 10 describe la motivación, definición y uso de vistas relacionales, junto con la especificación de definiciones de vistas para formularios e informes de captura de datos. El capítulo 11 presenta conceptos y prácticas de codificación de los lenguajes de programación de bases de datos, procedimientos almacenados y disparadores para la personalización de las aplicaciones de bases de datos.

La parte 6 abarca temas avanzados del desarrollo de bases de datos. El capítulo 12 ofrece una descripción del diseño e integración de vistas, los cuales son conceptos de modelado de datos para los esfuerzos que se realizan en el desarrollo de bases de datos más grandes. El capítulo 13 ofrece un estudio de caso completo que permite a los alumnos obtener información acerca de las

dificultades que se enfrentan al aplicar el diseño de bases de datos y las habilidades de desarrollo de aplicaciones para una base de datos empresarial del mundo real.

Más allá de las habilidades del diseño de bases de datos y desarrollo de aplicaciones, este libro prepara a los alumnos para que ejerzan carreras como especialistas de bases de datos. Los estudiantes necesitan comprender las responsabilidades, herramientas y procesos empleados por los administradores de datos y de bases de datos, así como otros ambientes diversos en los que operan las bases de datos.

Los capítulos de la parte 7 hacen énfasis en la función que desempeñan los especialistas de bases de datos y los detalles implicados en la administración de las mismas en diversos ambientes de operación. El capítulo 14 ofrece el contexto para otros capítulos por medio de una vasta cobertura de las responsabilidades, herramientas y procesos utilizados por los administradores de datos y de bases de datos. Los otros capítulos de la parte 4 proporcionan un fundamento para la administración de bases de datos en ambientes de importancia: el capítulo 15, sobre procesamiento de transacciones; el capítulo 16, en almacenes de datos; el capítulo 17, en procesos y datos distribuidos, y el capítulo 18 en administración de bases de datos orientadas a objetos. Estos capítulos hacen hincapié en conceptos, arquitecturas e importantes opciones de diseño para los especialistas de bases de datos.

## Enfoque del texto y tema

Para apoyar la adquisición de las habilidades necesarias para el aprendizaje y comprensión del desarrollo de aplicaciones, el diseño de bases de datos y la administración de las mismas, este libro se adhiere a tres principios guía:

1. *Combinación de conceptos y práctica.* La administración de las bases de datos se aprende con mayor facilidad cuando los conceptos están estrechamente ligados con la práctica del diseño e implementación de bases de datos con el uso de un DBMS comercial. El libro y los complementos que le acompañan han sido diseñados para proporcionar una estrecha integración entre los conceptos y la práctica por medio de las siguientes características:
  - Ejemplos de SQL para Access y Oracle, además de cobertura para SQL:2003.
  - Énfasis en la relación entre el desarrollo de aplicaciones y la formulación de consultas.
  - Uso de una notación de modelado de datos por medio de herramientas CASE profesionales y una herramienta académica de uso sencillo (ER Assistant).
  - Capítulos complementarios de prácticas de laboratorio que combinan los conceptos del libro con los detalles de los DBMS comerciales.
2. *Énfasis en habilidades para la solución de problemas.* Este libro presenta los lineamientos para la solución de problemas que ayudarán a los alumnos a dominar las habilidades fundamentales del modelado de datos, normalización, formulación de consultas y desarrollo de aplicaciones. El libro y los complementos asociados proporcionan una riqueza de preguntas, problemas, casos de estudio y prácticas de laboratorio en las que los alumnos pueden aplicar sus habilidades. Con el dominio de las habilidades esenciales, los estudiantes tendrán motivación para un mayor aprendizaje sobre bases de datos y un cambio en su manera de pensar la computación en general.
3. *Material introductorio y avanzado.* Los alumnos de negocios que empleen este libro pueden provenir de una gran variedad de entornos. Este libro les ofrece suficiente profundidad para satisfacer a los alumnos más ansiosos por aprender. No obstante, las partes avanzadas se colocan de manera tal que puedan evitarse por aquellos que no deseen cubrirlas.

## Características pedagógicas

Con el fin de ayudar a los alumnos a navegar a través del contenido capítular de una manera sistemática, este libro contiene las siguientes características pedagógicas:

- **Objetivos de aprendizaje** que se enfocan en el conocimiento y habilidades que los alumnos adquirirán al estudiar dicho capítulo.
- **Vistas generales** que proporcionan un vistazo a los contenidos capitulares.
- **Términos fundamentales** que son resaltados y definidos en los márgenes conforme aparecen en el capítulo.
- **Ejemplos** que son claramente separados del resto del material del capítulo para dar una revisión más sencilla y cubrir los propósitos de estudio.
- **Ejemplos de bases de datos** en funcionamiento —universidad y captura de órdenes— con iconos en los márgenes que llaman la atención del alumno hacia los ejemplos.
- **Reflexiones finales** que resumen el contenido capítular en relación con los objetivos de aprendizaje.
- **Revisión de conceptos** que son llamadas conceptuales importantes del capítulo, y no sólo una lista de terminología.
- **Preguntas** que se proporcionan para revisar los conceptos capitulares.
- **Problemas** que ayudan a que el alumno practique e implemente las habilidades detalladas que se presentaron en el capítulo.
- **Referencias adicionales para estudio** que resaltan las fuentes adicionales del contenido capítular para los alumnos.
- **Apéndices capitulares** que proporcionan detalles adicionales y resúmenes convenientes de ciertos principios o prácticas.

Al final del libro, los alumnos encontrarán los siguientes recursos adicionales:

- **Glosario:** Proporciona una lista completa de términos y definiciones empleados a lo largo del libro.
- **Bibliografía:** Un listado de material impreso útil de índole industrial y académico para investigación o estudio.

## Materiales de apoyo

---

Esta obra cuenta con interesantes complementos que fortalecen los procesos de enseñanza-aprendizaje, así como la evaluación de los mismos, los cuales se otorgan a profesores que adoptan este texto para sus cursos. Para obtener más información y conocer la política de entrega de estos materiales, contacte a su representante McGraw-Hill.

## Recorrido de aprendizaje

---

El libro puede cubrirse en diverso orden en una secuencia de uno o dos semestres. El autor ha enseñado un curso de un semestre con el siguiente orden: desarrollo de aplicaciones, desarrollo de bases de datos y ambientes de procesamiento de bases de datos. Este orden ofrece la ventaja de poder cubrir el material más concreto (desarrollo de aplicaciones) antes de un material más abstracto (desarrollo de bases de datos). Los capítulos de laboratorio y tareas se emplean para proporcionar mayores oportunidades de práctica, más allá de los ejemplos que aparecen en el libro. Para que pueda cubrirse en un semestre, los temas avanzados se saltan en los capítulos 8 y del 11 al 18.

Un segundo orden deberá cubrir el desarrollo de bases de datos antes de estudiar la aplicación de bases de datos. Para seguir con este orden, el autor recomienda seguir un orden con los capítulos del libro de la siguiente manera: 1, 2, 5, 6, 3, 7, 4, 9 y 10. El material sobre conversión de esquemas que se presenta en el capítulo 6 deberá cubrirse después del capítulo 3. Este orden apoya una cobertura más amplia del desarrollo de bases de datos, al tiempo que no rechaza el desarrollo de aplicaciones. Para poderse cubrir en un semestre, los temas avanzados se saltan en los capítulos 8 y del 11 al 18.

Un tercer orden posible es utilizar el texto en una secuencia de dos semestres. El primer semestre abarca lo fundamental en cuanto a la administración de las bases de datos, que aparece en las partes 1 y 2, modelado de datos y normalización de las partes 3 y 4, así como formulación avanzada de consultas de las partes 5 y 6. El segundo curso hace énfasis en las habilidades para la administración de bases de datos con el diseño físico de bases de datos de la parte 4, disparadores y procedimientos almacenados que aparecen en la parte 5 y ambientes de procesamiento de la parte 7, junto con material adicional sobre la administración de bases de datos empresariales. Un proyecto completo puede emplearse en el segundo curso para integrar el desarrollo de aplicaciones, desarrollo de bases de datos y administración de bases de datos.

## Recursos para el alumno

---

- **ER Assistant:** Herramienta de uso sencillo para modelado de datos que puede emplearse para dibujar y analizar ERD; disponible en [www.mhhe.com/mannino](http://www.mhhe.com/mannino)

## Reconocimientos

---

La tercera edición es la culminación de muchos años de trabajo. Antes de comenzar con la primera edición escribí tutoriales, prácticas de laboratorio y estudios de caso. El material fue utilizado primero como complemento a otros libros de texto. Después, motivado por mis alumnos, este material se empleó sin libro de texto. Este material, revisado en múltiples ocasiones a través de los comentarios de los alumnos, fue el fundamento para la primera edición. Durante el desarrollo de la primera edición, el material se evaluó en el salón de clases durante tres años con cientos de alumnos de nivel licenciatura y cursos superiores, junto con una revisión cuidadosa por medio de cuatro borradores hechos por muchos revisores externos. La segunda edición se realizó a través del uso de la primera edición durante tres años en el salón de clases, además de que el autor había impartido durante muchos años un curso avanzado de bases de datos. La tercera edición se desarrolló con la experiencia de tres años de uso de la segunda edición en cursos básicos y avanzados de bases de datos.

Deseo reconocer el excelente apoyo que recibí en el término de este proyecto. En primer lugar agradezco a mis muchos alumnos de bases de datos, especialmente a los de ISMG6080, ISMG6480 e ISMG4500 de la University of Colorado en Denver. Sus comentarios y reacción al libro de texto han sido invaluosables para su mejoría.

En segundo lugar, agradezco a los muchos revisores que proporcionaron retroalimentación a los diversos borradores de este libro de texto:

**Kirk P. Arnett**  
*Mississippi State University*

**Reza Barkhi**  
*Virginia Polytechnic Institute and State University*

**William Barnett**  
*University of Louisiana—Monroe*

**Jack D. Becker**  
*University of North Texas*

**Nabil Bedewi**  
*George Mason University*

**France Belanger**  
*Virginia Polytechnic Institute and State University*

**John Bradley**  
*East Carolina University*

**Susan Brown**  
*Indiana University—Bloomington*

**Debra L. Chapman**  
*University of South Alabama*

**Dr. Qiyang Chen**  
*Montclair State University*

**Amita G. Chin**  
*Virginia Commonwealth University*

**Russell Ching**  
*California State University—Sacramento*

**P. C. Chu**  
*The Ohio State University*

<b>Carey Cole</b> <i>James Madison University</i>	<b>Mary Malliaris</b> <i>Loyola University—Chicago</i>
<b>Erman Coskun</b> <i>Le Moyne College</i>	<b>Bruce McLaren</b> <i>Indiana State University</i>
<b>Connie W. Crook</b> <i>University of North Carolina—Charlotte</i>	<b>Dr. Kathryn J. Moland</b> <i>Livingstone College</i>
<b>Robert Louis Gilson</b> <i>Washington State University</i>	<b>Hossein Larry Najafi</b> <i>University of Wisconsin River Falls</i>
<b>Jian Guan</b> <i>University of Louisville</i>	<b>Karen S. Nantz</b> <i>Eastern Illinois University</i>
<b>Diane Hall</b> <i>Auburn University</i>	<b>Ann Nelson</b> <i>High Point University</i>
<b>Dr. Joseph T. Harder</b> <i>Indiana State University</i>	<b>Hamid Nemati</b> <i>University of North Carolina—Greensboro</i>
<b>Mark Hwang</b> <i>Central Michigan University</i>	<b>Robert Phillips</b> <i>Radford University</i>
<b>Balaji Janamanchi</b> <i>Texas Tech University</i>	<b>Lara Preiser-Houy</b> <i>California State Polytechnic University—Pomona</i>
<b>Nenad Jukic</b> <i>Loyola University Chicago</i>	<b>Young U. Ryu</b> <i>University of Texas—Dallas</i>
<b>Rajeev Kaula</b> <i>Southwest Missouri State University</i>	<b>Werner Schenk</b> <i>University of Rochester</i>
<b>Sung-kwan Kim</b> <i>University of Arkansas—Little Rock</i>	<b>Dr. Barbara A. Schuldt</b> <i>Southeastern Louisiana University</i>
<b>Yong Jin Kim</b> <i>SUNY Binghamton</i>	<b>Philip J. Scieme</b> <i>Dominican College</i>
<b>Barbara Klein</b> <i>University of Michigan—Dearborn</i>	<b>Richard S. Segall</b> <i>Arkansas State University</i>
<b>Constance Knapp</b> <i>Pace University</i>	<b>Hsueh-Chi Joshua Shih</b> <i>National Yunlin University of Science and Technology</i>
<b>Alexis Koster</b> <i>San Diego State University</i>	<b>Elizabeth Paige Sigman</b> <i>Georgetown University</i>
<b>Jean-Pierre Kuilboer</b> <i>University of Massachusetts—Boston</i>	<b>Vickee Stedham</b> <i>St. Petersburg College</i>
<b>Alan G. Labouseur</b> <i>Marist College</i>	<b>Jeffrey A. Stone</b> <i>Pennsylvania State University</i>
<b>Dr. William M. Lankford</b> <i>University of West Georgia</i>	<b>Dr. Thomas P. Sturm</b> <i>University of St. Thomas</i>
<b>Eitel Lauria</b> <i>Marist College</i>	<b>A. Tansel</b> <i>Baruch College—CUNY</i>
<b>Anita Lee-Post</b> <i>University of Kentucky</i>	<i>Bilkent University—Ankara, Turkey</i>
<b>John D. (Skip) Lees</b> <i>California State University—Chico</i>	<b>Sylvia Unwin</b> <i>Bellevue Community College</i>
<b>William Leigh</b> <i>University of Central Florida</i>	<b>Stuart Varden</b> <i>Pace University</i>
<b>Robert Little</b> <i>Auburn University—Montgomery</i>	<b>Santosh S. Venkatraman</b> <i>University of Arkansas—Little Rock</i>
<b>Dr. Jie Liu</b> <i>Western Oregon University</i>	

**F. Stuart Wells**

*Tennessee Technological University*

**Larry West**

*University of Central Florida*

**Hsui-lin Winkler**

*Pace University*

**Peter Wolcott**

*University of Nebraska—Omaha*

**James L. Woolley**

*Western Illinois University*

**Brian Zelli**

*SUNY Buffalo*

Sus comentarios, especialmente los de índole crítica, me han ayudado en gran medida para afinar los detalles de este libro.

En tercer lugar, deseo expresar mi agradecimiento a los editores de la versión en inglés, Paul Ducham y Liz Farina, por su guía en este proceso, así como a Jim Labeots, Kami Cartes y a los demás amigos de McGraw-Hill, quienes me ayudaron en la producción y publicación de este texto. Finalmente, agradezco a mi esposa, Monique, por su ayuda con el libro y sus complementos, así como por su apoyo moral a mi esfuerzo.

*Michael V. Mannino*

# Contenido breve

---

## PARTE UNO

### Introducción a los ambientes de base de datos 1

- 1 Introducción a la administración de base de datos 3
- 2 Introducción al desarrollo de bases de datos 23

## PARTE DOS

### Comprendiendo las bases de datos relacionales 43

- 3 El modelo relacional de datos 45
- 4 Formulación de consultas con SQL 79

## PARTE TRES

### Modelado de datos 133

- 5 Comprensión de los diagramas de entidad-relación 135
- 6 Desarrollo de modelo de datos para bases de datos de negocios 167

## PARTE CUATRO

### Diseño de bases de datos relacionales 217

- 7 Normalización de tablas relacionales 219
- 8 Diseño físico de bases de datos 249

## PARTE CINCO

### Desarrollo de aplicaciones con bases de datos relacionales 295

- 9 Formulación avanzada de consultas con SQL 297
- 10 Desarrollo de aplicaciones con vistas 339
- 11 Procedimientos almacenados y disparadores 375

## PARTE SEIS

### Desarrollo avanzado de bases de datos 425

- 12 Diseño e integración de vistas 427
- 13 Desarrollo de base de datos para Student Loan Limited 449

## PARTE SIETE

### Administración de entornos de bases de datos 479

- 14 Administración de datos y bases de datos 481
- 15 Administración de transacciones 515
- 16 Tecnología y administración de data warehouse 553
- 17 Procesamiento cliente-servidor, procesamiento de bases de datos paralelas y bases de datos distribuidas 605
- 18 Sistemas de administración de bases de datos de objetos 641

## GLOSARIO 679

## BIBLIOGRAFÍA 696

## ÍNDICE 698

# Contenido

---

## PARTE UNO

### INTRODUCCIÓN A LOS AMBIENTES DE BASE DE DATOS 1

#### Capítulo 1

##### Introducción a la administración de base de datos 3

Objetivos de aprendizaje 3

Panorama general 3

1.1 Características de la base de datos 4

1.2 Características de los sistemas de administración de bases de datos 6

1.2.1 Definición de base de datos 6

1.2.2 Acceso no procedural 8

1.2.3 Desarrollo de aplicaciones e interfase del lenguaje procedural 9

1.2.4 Funciones de soporte para las operaciones de base de datos 10

1.2.5 Funciones de terceros 11

1.3 Desarrollo de la tecnología de base de datos y la estructura de mercado 11

1.3.1 Evolución de la tecnología de bases de datos 12

1.3.2 El mercado actual del software de base de datos 13

1.4 Arquitecturas de los sistemas de administración de bases de datos 14

1.4.1 Independencia de datos y la arquitectura de los tres esquemas 14

1.4.2 Procesamiento distribuido y la arquitectura cliente-servidor 16

1.5 Impactos organizacionales de la tecnología de base de datos 17

1.5.1 Interactuando con las bases de datos 17

1.5.2 Administración de recursos de información 19

Reflexión final 20

Revisión de conceptos 20

Preguntas 21

Problemas 22

Referencias para ampliar su estudio 22

#### Capítulo 2

##### Introducción al desarrollo de bases de datos 23

Objetivos de aprendizaje 23

Panorama general 23

2.1 Sistemas de información 23

2.1.1 Componentes de los sistemas de información 24

2.1.2 Proceso del desarrollo de sistemas de información 25

2.2 Objetivos del desarrollo de base de datos 26

2.2.1 Desarrollar un vocabulario común 27

2.2.2 Definición del significado de los datos 27

2.2.3 Asegurar la calidad de los datos 27

2.2.4 En busca de la implementación eficiente 28

2.3 Proceso de desarrollo de la base de datos 28

2.3.1 Fases del desarrollo de base de datos 29

2.3.2 Habilidades en el desarrollo de bases de datos 32

2.4 Herramientas para el desarrollo de bases de datos 34

2.4.1 Diagramación 35

2.4.2 Documentación 35

2.4.3 Análisis 35

2.4.4 Herramientas de prototipos 36

2.4.5 Herramientas CASE comerciales 37

Reflexión final 39

Revisión de conceptos 39

Preguntas 40

Problemas 41

Referencias para ampliar su estudio 41

## PARTE DOS

### COMPRENDIENDO LAS BASES DE DATOS RELACIONALES 43

#### Capítulo 3

##### El modelo relacional de datos 45

Objetivos de aprendizaje 45

Panorama general 45

3.1 Elementos básicos 46

3.1.1 Tablas 46

3.1.2 Conexiones entre tablas 47

3.1.3 Terminología alternativa 49

3.2 Reglas de integridad 49

3.2.1 Definición de las reglas de integridad 49

3.2.2 Aplicación de las reglas de integridad 50

3.2.3 Representación gráfica de la integridad referencial 53

3.3 Acciones para eliminar y actualizar filas referenciadas 54

3.4 Operadores de álgebra relacional 56

3.4.1 Operadores de restricción (select) y proyección (project) 56

3.4.2 Operador de producto cruz extendido 57

3.4.3 Operador de enlace (join) 59

3.4.4 Operador de enlace externo (outer join) 61

3.4.5 Operadores de unión, intersección y diferencia 63

3.4.6 Operador resumir (summarize) 65

3.4.7 Operador dividir (divide) 66

3.4.8 Resumen de operadores 68

Reflexión final 68

Revisión de conceptos 69

Preguntas 69

Problemas 70

Referencias para ampliar su estudio 73

**Apéndice 3.A**

**Sentencias CREATE TABLE para la base de datos de la universidad** 73

**Apéndice 3.B**

**Resumen de sintaxis en SQL:2003** 74

**Apéndice 3.C**

**Generación de valores únicos para llaves primarias** 76

**Capítulo 4****Formulación de consultas con SQL** 79

Objetivos de aprendizaje 79

Panorama general 79

**4.1 Antecedentes** 80

4.1.1 Breve historia de SQL 80

4.1.2 Alcance de SQL 81

**4.2 Empecemos con la sentencia SELECT** 82

4.2.1 Problemas de tabla simple 84

4.2.2 Tablas enlazadas (joining) 89

4.2.3 Resumen de tablas con GROUP BY y HAVING 91

4.2.4 Mejorar el formato de los resultados 95

**4.3 Proceso de evaluación conceptual para las sentencias SELECT** 97**4.4 Preguntas críticas para la generación de consultas** 101**4.5 Mejorando las habilidades de generación de consultas mediante ejemplos** 103

4.5.1 Enlaces de tablas múltiples con el estilo de producto cruz 103

4.5.2 Enlazando tablas múltiples con el estilo del operador de enlace 106

4.5.3 Autoenlaces (self-joins) y enlaces múltiples entre dos tablas 109

4.5.4 Combinando enlaces y agrupación 110

4.5.5 Operadores de conjuntos tradicionales en SQL 111

**4.6 Sentencias de modificación de SQL** 113

Reflexión final 115

Revisión de conceptos 116

Preguntas 119

Problemas 120

Referencias para ampliar su estudio 127

**Apéndice 4.A**

**Resumen de sintaxis de SQL:2003** 128

**Apéndice 4.B**

**Diferencias de sintaxis entre los principales productos DBMS** 131

**PARTE TRES****MODELADO DE DATOS** 133**Capítulo 5****Comprepción de los diagramas de entidad-relación** 135

Objetivos de aprendizaje 135

Panorama general 135

**5.1 Introducción a los diagramas de entidad-relación** 136

5.1.1 Símbolos básicos 136

5.1.2 Cardinalidad de las relaciones 137

5.1.3 Comparación con los diagramas de bases de datos relacionales 140

**5.2 Comprensión de las relaciones** 141

5.2.1 Identificación de dependencias (entidades débiles y relación identificable) 141

5.2.2 Patrones de relación 142

5.2.3 Equivalencia entre las relaciones 1-M y M-N 146

**5.3 Clasificación en el modelo de entidad-relación** 147

5.3.1 Jerarquías de generalización 148

5.3.2 Restricciones de separación e integridad 148

5.3.3 Niveles múltiples de generalización 149

**5.4 Resumen de notación y reglas de diagramación** 150

5.4.1 Resumen de notación 150

5.4.2 Reglas de diagramación 152

**5.5 Comparación a otras notaciones** 156

5.5.1 Variaciones ERD 156

5.5.2 Notación del diagrama de clases del lenguaje de modelado unificado 157

Reflexión final 159

Revisión de conceptos 160

Preguntas 160

Problemas 161

Referencias para ampliar su estudio 166

**Capítulo 6****Desarrollo de modelo de datos para bases de datos de negocios** 167

Objetivos de aprendizaje 167

Panorama general 167

**6.1 Análisis de problemas de modelado de datos para negocios** 168

6.1.1 Guías para analizar las necesidades de información de los negocios 168

6.1.2 Análisis de los requerimientos de información para la base de datos de la compañía de servicios de agua 171

**6.2 Refinamientos a un ERD** 173

6.2.1 Transformación de los atributos en tipos de entidad 173

6.2.2 División de atributos compuestos 173

6.2.3 Expansión de los tipos de entidad 173

6.2.4 Transformación de una entidad débil en una entidad fuerte 174

6.2.5 Agregando historia 175

6.2.6 Añadiendo jerarquías de generalización 177

6.2.7 Resumen de transformaciones 178

**6.3 Finalización de un ERD** 179

6.3.1 Documentación de un ERD 179

6.3.2 Detección de errores comunes de diseño 181

**6.4 Conversión de un ERD en tablas relacionales** 183

6.4.1 Reglas básicas de conversión 183

6.4.2 Conversión de relaciones opcionales 1-M 188

6.4.3 Conversión de jerarquías de generalización 190

- 6.4.4 *Conversión de relaciones 1-1* 191
- 6.4.5 *Ejemplo de conversión comprensiva* 193
- Reflexión final 195
- Revisión de conceptos 196
- Preguntas 196
- Problemas 197
- Referencias para ampliar su estudio 215

## **PARTE CUATRO**

### **DISEÑO DE BASES DE DATOS RELACIONALES 217**

#### **Capítulo 7**

##### **Normalización de tablas relacionales 219**

- Objetivos de aprendizaje 219
- Panorama general 219
- 7.1 Panorama general del diseño de bases de datos relacionales 220
  - 7.1.1 *Evite anomalías de modificación* 220
  - 7.1.2 *Dependencias funcionales* 221
- 7.2 Formas normales 223
  - 7.2.1 *Primera forma normal* 224
  - 7.2.2 *Segunda y tercera forma normal* 224
  - 7.2.3 *Fórmula normal de Boyce-Codd* 227
  - 7.2.4 *Procedimiento de síntesis simple* 229
- 7.3 Refinamiento de las relaciones M-way 232
  - 7.3.1 *Independencia de relación* 232
  - 7.3.2 *Dependencias multivaluadas y cuarta forma normal* 235
- 7.4 Formas normales de alto nivel 236
  - 7.4.1 *Quinta forma normal* 236
  - 7.4.2 *Forma normal de llave de dominio* 237
- 7.5 Cuestiones prácticas acerca de normalización 237
  - 7.5.1 *Función de la normalización en el proceso de desarrollo de base de datos* 237
  - 7.5.2 *Análisis del objetivo de normalización* 238
- Reflexión final 238
- Revisión de conceptos 239
- Preguntas 239
- Problemas 240
- Referencias para ampliar su estudio 248

#### **Capítulo 8**

##### **Diseño físico de bases de datos 249**

- Objetivos de aprendizaje 249
- Panorama general 249
- 8.1 Panorama general del diseño físico de bases de datos 250
  - 8.1.1 *Niveles de almacenamiento de las bases de datos* 250
  - 8.1.2 *Objetivos y restricciones* 251
  - 8.1.3 *Entradas, salidas y entorno* 252
  - 8.1.4 *Dificultades* 253
- 8.2 Entradas del diseño físico de bases de datos 253
  - 8.2.1 *Perfiles de las tablas* 253
  - 8.2.2 *Perfiles de la aplicación* 255
- 8.3 Estructuras de archivos 256
  - 8.3.1 *Archivos secuenciales* 256

- 8.3.2 *Archivos hash* 257
- 8.3.3 *Archivos de árbol multiforme (Btrees)* 259
- 8.3.4 *Índices bitmap* 266
- 8.3.5 *Resumen de estructuras de archivos* 267
- 8.4 Optimización de consultas 268
  - 8.4.1 *Tareas de traducción* 268
  - 8.4.2 *Mejoras en las decisiones de optimización* 271
- 8.5 Selección de índices 274
  - 8.5.1 *Definición del problema* 274
  - 8.5.2 *Equilibrios y dificultades* 276
  - 8.5.3 *Reglas de selección* 277
- 8.6 Opciones adicionales del diseño físico de base de datos 280
  - 8.6.1 *Desnormalización* 280
  - 8.6.2 *Formateo de registros* 282
  - 8.6.3 *Procesamiento paralelo* 283
  - 8.6.4 *Otras formas de mejorar el desempeño* 284
- Reflexión final 285
- Revisión de conceptos 285
- Preguntas 286
- Problemas 287
- Referencias para ampliar su estudio 293

## **PARTE CINCO**

### **DESARROLLO DE APLICACIONES CON BASES DE DATOS RELACIONALES 295**

#### **Capítulo 9**

##### **Formulación avanzada de consultas con SQL 297**

- Objetivos de aprendizaje 297
- Panorama general 297
- 9.1 Problemas de enlace externo (outer join) 298
  - 9.1.1 *Soporte de SQL para problemas de enlace externo* 298
  - 9.1.2 *Combinación de enlaces internos y externos* 301
- 9.2 Entendiendo las consultas empaquetadas 303
  - 9.2.1 *Consultas empaquetadas tipo I* 303
  - 9.2.2 *Formulaciones SQL limitadas para problemas de diferencia* 305
  - 9.2.3 *Uso de consultas empaquetadas tipo II para problemas de diferencia* 308
  - 9.2.4 *Consultas empaquetadas con la cláusula FROM* 312
- 9.3 Formulación de problemas de división 314
  - 9.3.1 *Revisión del operador dividir* 314
  - 9.3.2 *Problemas sencillos de división* 315
  - 9.3.3 *Problemas de división avanzados* 317
- 9.4 Consideraciones de valor nulo 320
  - 9.4.1 *Efecto en condiciones simples* 320
  - 9.4.2 *Efecto sobre las condiciones compuestas* 321
  - 9.4.3 *Efecto sobre los cálculos conjuntos y la agrupación* 323
- Reflexión final 324
- Revisión de conceptos 325
- Preguntas 327
- Problemas 328
- Referencias para ampliar su estudio 332

<b>Apéndice 9.A</b>	
<b>Uso de sentencias múltiples en Microsoft Access</b>	<b>332</b>
<b>Apéndice 9.B</b>	
<b>Resumen de la sintaxis de SQL:2003</b>	<b>333</b>
<b>Apéndice 9.C</b>	
<b>Anotación de Oracle 8i para enlaces externos</b>	<b>335</b>
 <b>Capítulo 10</b>	
<b>Desarrollo de aplicaciones con vistas</b> <b>339</b>	
Objetivos de aprendizaje 339	
Panorama general 339	
<b>10.1</b>	Antecedentes 340
10.1.1	<i>Motivación</i> 340
10.1.2	<i>Definición de vista</i> 340
<b>10.2</b>	Uso de vistas para recuperación 342
10.2.1	<i>Uso de vistas en enunciados SELECT</i> 342
10.2.2	<i>Procesamiento de consultas con referencias a vista</i> 344
<b>10.3</b>	Actualización con el uso de vistas 346
10.3.1	<i>Vistas de actualización con tabla única</i> 346
10.3.2	<i>Vistas de actualización y tabla múltiple</i> 349
<b>10.4</b>	Uso de vistas en formularios jerárquicos 353
10.4.1	<i>¿Qué es un formulario jerárquico?</i> 353
10.4.2	<i>Relación entre formularios jerárquicos y tablas</i> 354
10.4.3	<i>Habilidades de formulación de consultas para formularios jerárquicos</i> 355
<b>10.5</b>	Uso de vistas en los reportes 359
10.5.1	<i>¿Qué es un reporte jerárquico?</i> 359
10.5.2	<i>Habilidades para la formulación de consultas en reportes jerárquicos</i> 361
Reflexión final 362	
Revisión de conceptos 362	
Preguntas 363	
Problemas 364	
Referencias para ampliar su estudio 371	
<b>Apéndice 10.A</b>	
<b>Resumen de sintaxis de SQL: 2003</b>	<b>372</b>
<b>Apéndice 10.B</b>	
<b>Reglas para vistas enlazadas que se pueden actualizar en Oracle</b> <b>372</b>	
 <b>Capítulo 11</b>	
<b>Procedimientos almacenados y disparadores</b> <b>375</b>	
Objetivos de aprendizaje 375	
Panorama general 375	
<b>11.1</b>	Lenguajes de programación de bases de datos y PL/SQL 376
11.1.1	<i>Motivación para los lenguajes de programación de base de datos</i> 376
11.1.2	<i>Aspectos de diseño</i> 378
11.1.3	<i>Sentencias PL/SQL</i> 380
11.1.4	<i>Ejecución de las sentencias PL/SQL en bloques anónimos</i> 386
<b>11.2</b>	Procedimientos almacenados 388
11.2.1	<i>Procedimientos PL/SQL</i> 389
11.2.2	<i>Funciones PL/SQL</i> 392
<b>11.3</b>	Disparadores ( <i>triggers</i> ) 402
11.3.1	<i>Motivación y clasificación de los disparadores</i> 402
11.3.2	<i>Disparadores de Oracle</i> 403
11.3.3	<i>Cómo entender la ejecución de un disparador</i> 414
Reflexión final 417	
Revisión de conceptos 417	
Preguntas 419	
Problemas 420	
Referencias para ampliar su estudio 422	
<b>Apéndice 11.A</b>	
<b>Resumen de la sintaxis de SQL:2003</b>	<b>423</b>
 <b>PARTE SEIS</b>	
<b>DESARROLLO AVANZADO DE BASES DE DATOS</b> <b>425</b>	
 <b>Capítulo 12</b>	
<b>Diseño e integración de vistas</b> <b>427</b>	
Objetivos de aprendizaje 427	
Panorama general 427	
<b>12.1</b>	Motivación para el diseño e integración de vistas 428
<b>12.2</b>	Diseño de vistas con formularios 429
12.2.1	<i>Análisis de formularios</i> 429
12.2.2	<i>Análisis de las relaciones M-way utilizando formularios</i> 435
<b>12.3</b>	Integración de vistas 439
12.3.1	<i>Enfoque de integración incremental y paralelo</i> 439
12.3.2	<i>Ejemplos de integración de vistas</i> 442
Reflexión final 444	
Revisión de conceptos 444	
Preguntas 445	
Problemas 445	
Referencias para ampliar su estudio 447	
 <b>Capítulo 13</b>	
<b>Desarrollo de base de datos para Student Loan Limited</b> <b>449</b>	
Objetivos de aprendizaje 449	
Panorama general 449	
<b>13.1</b>	Descripción del caso 450
13.1.1	<i>Panorama general</i> 450
13.1.2	<i>Flujo de trabajo</i> 450
<b>13.2</b>	Modelado conceptual de datos 455
13.2.1	<i>ERD para el formulario de origen del préstamo</i> 455
13.2.2	<i>Integración incremental después de agregar una carta de declaración</i> 455
13.2.3	<i>Integración incremental después de agregar el estado de cuenta</i> 458
13.2.4	<i>Integración incremental después de agregar el reporte de actividad del préstamo</i> 459

<b>13.3</b>	Refinamiento del esquema conceptual 461
13.3.1	<i>Conversión del esquema</i> 461
13.3.2	<i>Normalización</i> 462
<b>13.4</b>	Diseño físico de la base de datos y desarrollo de aplicaciones 464
13.4.1	<i>Perfiles de aplicaciones y tablas</i> 464
13.4.2	<i>Selección de índices</i> 465
13.4.3	<i>Datos derivados y decisiones de desnormalización</i> 466
13.4.4	<i>Otras decisiones de implementación</i> 467
13.4.5	<i>Desarrollo de aplicaciones</i> 467
Reflexión final	469
Revisión de conceptos	470
Preguntas	470
Problemas	471
<b>Apéndice 13.A</b>	
<b>Glosario de los campos del formulario y reporte</b>	<b>472</b>
<b>Apéndice 13.B</b>	
<b>Enunciados CREATE TABLE</b>	<b>474</b>

## PARTE SIETE

### ADMINISTRACIÓN DE ENTORNOS DE BASES DE DATOS 479

#### Capítulo 14

#### Administración de datos y bases de datos 481

Objetivos de aprendizaje	481
Panorama general	481
<b>14.1</b>	Contexto organizacional para administrar bases de datos 482
14.1.1	<i>Apoyo de las bases de datos en la toma de decisiones administrativas</i> 482
14.1.2	<i>Administración de los recursos de información para el manejo del conocimiento</i> 483
14.1.3	<i>Responsabilidades de los administradores de datos y los administradores de bases de datos</i> 484
<b>14.2</b>	Herramientas de administración de bases de datos 485
14.2.1	<i>Seguridad</i> 486
14.2.2	<i>Restricciones de integridad</i> 490
14.2.3	<i>Administración de disparadores y procedimientos almacenados</i> 493
14.2.4	<i>Manipulación del diccionario de datos</i> 495
<b>14.3</b>	Procesos para especialistas en bases de datos 497
14.3.1	<i>Planeación de datos</i> 497
14.3.2	<i>Selección y evaluación de los sistemas de administración de bases de datos</i> 498
<b>14.4</b>	Administración de entornos de bases de datos 503
14.4.1	<i>Procesamiento de transacciones</i> 503
14.4.2	<i>Procesamiento de data warehouses</i> 503
14.4.3	<i>Entornos distribuidos</i> 504
14.4.4	<i>Administración de bases de datos de objetos</i> 505
Reflexión final	505
Revisión de conceptos	506

Preguntas	508
Problemas	509
Referencias para ampliar su estudio	511

<b>Apéndice 14.A</b>	
<b>Resumen de la sintaxis SQL: 2003</b>	<b>511</b>

#### Capítulo 15

#### Administración de transacciones 515

Objetivos de aprendizaje	515
Panorama general	515
<b>15.1</b>	Aspectos básicos de las transacciones de bases de datos 516
15.1.1	<i>Ejemplos de transacciones</i> 516
15.1.2	<i>Propiedades de la transacción</i> 518
<b>15.2</b>	Control de la concurrencia 519
15.2.1	<i>Objetivo del control de concurrencia</i> 520
15.2.2	<i>Problemas de interferencia</i> 520
15.2.3	<i>Herramientas de control de concurrencia</i> 522
<b>15.3</b>	Administración de recuperaciones 526
15.3.1	<i>Dispositivos de almacenamiento de datos y tipos de fallas</i> 526
15.3.2	<i>Herramientas de recuperación</i> 527
15.3.3	<i>Procesos de recuperación</i> 529
<b>15.4</b>	Aspectos del diseño de transacciones 533
15.4.1	<i>Límite de transacción y puntos decisivos</i> 533
15.4.2	<i>Niveles de aislamiento</i> 536
15.4.3	<i>Momento de imposición de la restricción de integridad</i> 537
15.4.4	<i>Puntos de protección</i> 539
<b>15.5</b>	Administración del flujo de trabajo 539
15.5.1	<i>Caracterización de los flujos de trabajo</i> 540
15.5.2	<i>Tecnologías que permiten el flujo de trabajo</i> 540
Reflexión final	542
Revisión de conceptos	543
Preguntas	544
Problemas	545
Referencias para ampliar su estudio	551

<b>Apéndice 15.A</b>	
<b>Resumen de sintaxis de SQL:2003</b>	<b>551</b>

#### Capítulo 16

#### Tecnología y administración de data warehouse 553

Objetivos de aprendizaje	553
Panorama general	553
<b>16.1</b>	Conceptos básicos 554
16.1.1	<i>Procesamiento de transacciones versus apoyo a las decisiones</i> 554
16.1.2	<i>Características de los data warehouse</i> 554
16.1.3	<i>Arquitecturas para data warehouse</i> 556
16.1.4	<i>Minería de datos</i> 558
16.1.5	<i>Aplicaciones de los data warehouse</i> 559
<b>16.2</b>	Representación multidimensional de los datos 560
16.2.1	<i>Ejemplo de un cubo de datos multidimensional</i> 560
16.2.2	<i>Terminología multidimensional</i> 562

<p>16.2.3 <i>Datos de serie de tiempo</i> 564</p> <p>16.2.4 <i>Operaciones del cubo de datos</i> 564</p> <p><b>16.3 Soporte de DBMS relacional para data warehouse</b> 567</p> <p>16.3.1 <i>Modelado de datos relacionales para datos multidimensionales</i> 567</p> <p>16.3.2 <i>Representación de dimensión</i> 571</p> <p>16.3.3 <i>Extensiones para la cláusula GROUP BY para datos multidimensionales</i> 574</p> <p>16.3.4 <i>Vistas materializadas y reescritura de consultas</i> 583</p> <p>16.3.5 <i>Tecnologías de almacenamiento y optimización</i> 589</p> <p><b>16.4 Mantenimiento de un data warehouse</b> 591</p> <p>16.4.1 <i>Fuentes de datos</i> 591</p> <p>16.4.2 <i>Flujo de trabajo para el mantenimiento de un data warehouse</i> 592</p> <p>16.4.3 <i>Administración del proceso de refrescamiento</i> 594</p> <p>Reflexión final 596</p> <p>Revisión de conceptos 596</p> <p>Preguntas 597</p> <p>Problemas 598</p> <p>Referencias para ampliar su estudio 603</p>	<p><b>17.6 Procesamiento de bases de datos distribuidas</b> 631</p> <p>17.6.1 <i>Procesamiento de consulta distribuida</i> 631</p> <p>17.6.2 <i>Procesamiento de transacción distribuida</i> 632</p> <p>Reflexión final 635</p> <p>Revisión de conceptos 636</p> <p>Preguntas 636</p> <p>Problemas 638</p> <p>Referencias para ampliar su estudio 639</p>
--	---

## Capítulo 17

### Procesamiento cliente-servidor, procesamiento de bases de datos paralelas y bases de datos distribuidas 605

Objetivos de aprendizaje 605  
Panorama general 605

<p><b>17.1 Panorama del procesamiento distribuido y de los datos distribuidos</b> 606</p> <p>17.1.1 <i>Motivación para el procesamiento cliente-servidor</i> 606</p> <p>17.1.2 <i>Motivación para el procesamiento de bases de datos paralelas</i> 607</p> <p>17.1.3 <i>Motivación para datos distribuidos</i> 608</p> <p>17.1.4 <i>Resumen de ventajas y desventajas</i> 609</p> <p><b>17.2 Arquitecturas de bases de datos cliente-servidor</b> 609</p> <p>17.2.1 <i>Conflictos de diseño</i> 609</p> <p>17.2.2 <i>Descripción de arquitecturas</i> 611</p> <p><b>17.3 Procesamiento de bases de datos paralelas</b> 615</p> <p>17.3.1 <i>Conflictos de arquitecturas y diseño</i> 616</p> <p>17.3.2 <i>Tecnología comercial de bases de datos paralelas</i> 617</p> <p><b>17.4 Arquitecturas para sistemas de gestión de bases de datos distribuidas</b> 620</p> <p>17.4.1 <i>Arquitectura de componente</i> 620</p> <p>17.4.2 <i>Arquitecturas de esquema</i> 622</p> <p><b>17.5 Transparencia para procesamiento de bases de datos distribuidas</b> 624</p> <p>17.5.1 <i>Ejemplo motivacional</i> 624</p> <p>17.5.2 <i>Transparencia de fragmentación</i> 626</p> <p>17.5.3 <i>Transparencia de ubicación</i> 627</p> <p>17.5.4 <i>Transparencia de mapeo local</i> 628</p> <p>17.5.5 <i>Transparencia en bases de datos distribuidas de Oracle</i> 628</p>
--

<p><b>18.1 Motivación para la administración de bases de datos de objetos</b> 642</p> <p>18.1.1 <i>Datos complejos</i> 642</p> <p>18.1.2 <i>Desajuste en el sistema de tipos</i> 642</p> <p>18.1.3 <i>Ejemplos de aplicación</i> 643</p> <p><b>18.2 Principios orientados a objetos</b> 644</p> <p>18.2.1 <i>Encapsulación</i> 644</p> <p>18.2.2 <i>Herencia</i> 646</p> <p>18.2.3 <i>Polimorfismo</i> 647</p> <p>18.2.4 <i>Lenguajes de programación versus DBMS</i> 649</p> <p><b>18.3 Arquitecturas para la administración de bases de datos de objetos</b> 649</p> <p>18.3.1 <i>Objetos grandes y software externo</i> 650</p> <p>18.3.2 <i>Servidores especializados en medios</i> 650</p> <p>18.3.3 <i>Middleware de bases de datos de objetos</i> 651</p> <p>18.3.4 <i>Sistemas de administración de bases de datos relacionales de objetos para tipos definidos por el usuario</i> 652</p> <p>18.3.5 <i>Sistemas de administración de bases de datos orientadas a objetos</i> 654</p> <p>18.3.6 <i>Resumen de arquitecturas de bases de datos de objetos</i> 655</p> <p><b>18.4 Características de bases de datos de objetos en SQL:2003</b> 655</p> <p>18.4.1 <i>Tipos definidos por el usuario</i> 656</p> <p>18.4.2 <i>Definiciones de tabla</i> 658</p> <p>18.4.3 <i>Familias de subtablas</i> 661</p> <p>18.4.4 <i>Manipulación de objetos complejos y familias de subtablas</i> 662</p> <p><b>18.5 Características de base de datos de objetos en Oracle 10g</b> 664</p> <p>18.5.1 <i>Definición de tipos definidos por el usuario y tablas tipo en Oracle 10g</i> 665</p> <p>18.5.2 <i>Uso de tablas de tipos en Oracle 10g</i> 668</p> <p>18.5.3 <i>Otras características de objeto en Oracle 10g</i> 670</p> <p>Reflexión final 672</p> <p>Revisión de conceptos 673</p> <p>Preguntas 673</p> <p>Problemas 674</p> <p>Referencias para ampliar su estudio 678</p>
--

## Glosario 679

## Bibliografía 696

## Índice 698

Parte

# Introducción a los ambientes de base de datos

---

1

La parte 1 proporciona los antecedentes para el posterior estudio detallado del diseño, desarrollo de aplicaciones y administración de base de datos. Los capítulos de la parte 1 presentan los principios de la administración de base de datos y la naturaleza de su proceso de desarrollo. El capítulo 1 abarca los conceptos básicos de la administración de base de datos, incluyendo características, funciones y arquitecturas de sus sistemas de administración, su mercado y los impactos organizacionales de su tecnología. El capítulo 2 es una introducción en el contexto, los objetivos, las fases y las herramientas del proceso de desarrollo de una base de datos.

---

**Capítulo 1.** Introducción a la administración de base de datos

**Capítulo 2.** Introducción al desarrollo de bases de datos



# Capítulo 1

---

# Introducción a la administración de base de datos

## Objetivos de aprendizaje

Este capítulo es una introducción a la tecnología de las bases de datos y al impacto que tiene sobre las organizaciones. Al finalizar este capítulo el estudiante debe haber adquirido los siguientes conocimientos y habilidades:

- Describir las características de las bases de datos de negocios y las características de sus sistemas de administración.
- Comprender la importancia de los accesos no procedurales para obtener productividad del software.
- Apreciar los avances en la tecnología de base de datos y sus contribuciones a la sociedad moderna.
- Comprender el impacto de las arquitecturas de los sistemas de administración de base de datos en el procesamiento distribuido y en el mantenimiento del software.
- Percibir las oportunidades profesionales relacionadas con el desarrollo de aplicaciones de base de datos y con su administración.

## Panorama general

---

Quizás usted no se haya dado cuenta, pero su vida se ve afectada de forma dramática por la tecnología de base de datos. Las bases de datos computarizadas son vitales para el funcionamiento de las organizaciones modernas. Usted está en contacto con las bases de datos diariamente a través de actividades como comprar en el supermercado, retirar efectivo de un cajero automático, ordenar un libro en línea y registrarse en una clase. Las comodidades de su vida diaria, en parte, se deben a la proliferación de las bases de datos computarizadas y a su tecnología.

La tecnología de bases de datos no solamente mejora las operaciones diarias de las organizaciones, sino también la calidad de las decisiones que afectan nuestras vidas. Las bases de datos contienen un flujo de datos acerca de muchos aspectos de nuestras vidas: preferencias de consumo, uso de telecomunicaciones, historial crediticio, hábitos al ver la televisión, etc. La tecnología de base de datos ayuda a resumir este volumen de datos en información útil para la toma de decisiones. Los directivos utilizan la información obtenida para la toma de decisiones a largo plazo, como invertir en plantas y equipo, ubicar tiendas, agregar elementos nuevos al inventario e iniciar nuevos negocios.

Este primer capítulo marca el punto inicial para la exploración de la tecnología de base de datos: analiza las características de una base de datos, de un sistema de administración de base de datos, de arquitecturas de sistemas y el papel de las personas en la administración y uso de las mismas. El capítulo 2 proporciona un panorama conceptual del proceso de desarrollo de una base de datos. Este capítulo presenta una amplia imagen de la tecnología de base de datos y comparte la emoción del trabajo por venir.

## 1.1 Características de la base de datos

---

### **base de datos**

una colección de datos persistentes que pueden compartirse e interrelacionarse.

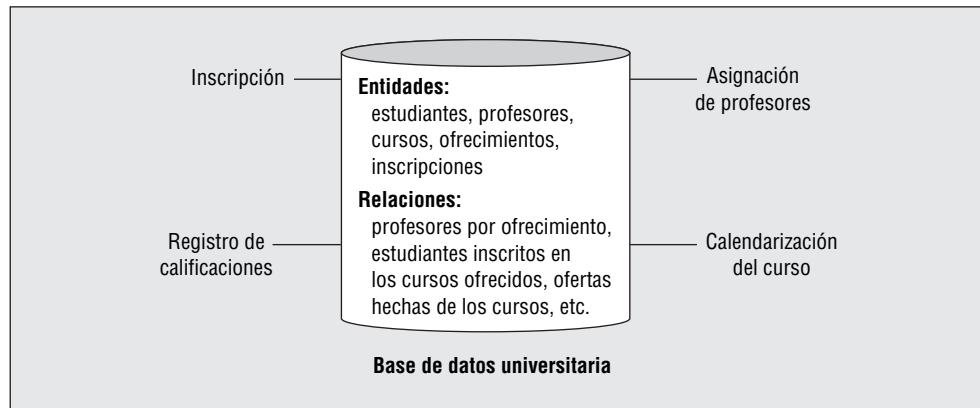
Las empresas reúnen todos los días montañas de hechos acerca de personas, cosas y eventos, como los números de tarjetas de crédito, estados de cuenta bancarios y montos de las compras. Las bases de datos contienen este tipo de hechos simples así como hechos no convencionales, tales como fotografías, huellas digitales, videos de productos y resúmenes de libros. Con la proliferación de Internet y de los medios para capturar datos de forma computarizada tenemos a nuestra disposición una gran cantidad de datos al hacer clic con el botón del ratón. Organizar estos datos para facilitar su consulta y mantenimiento es complicado. Por lo tanto, la administración de base de datos se ha convertido en una tarea vital en muchas organizaciones.

Antes de aprender sobre la administración de base de datos, primero debemos comprender algunas de sus propiedades más importantes, como las que se muestran en la siguiente lista:

- Persistente significa que los datos residen en un almacenamiento estable, tal como un disco magnético. Las organizaciones necesitan, por ejemplo, conservar los datos de sus clientes, proveedores e inventario en un almacenamiento estable, ya que se usan de forma continua. Una variable de un programa de computadora no es persistente porque reside en la memoria principal y desaparece después de que el programa finaliza. Persistencia no significa que los datos existan eternamente; cuando dejan de ser relevantes (por ejemplo, cuando un proveedor no continúa en el negocio), se eliminan o se archivan.
- La persistencia depende de la importancia del uso deseado. Por ejemplo, es importante conservar el kilometraje que usted recorre hasta su trabajo en caso de que sea autoempleado. De igual forma, el monto de sus gastos médicos es importante si puede deducirlo o si tiene una cuenta de gastos médicos. Ya que el almacenamiento y el mantenimiento de los datos es costoso, sólo deben almacenarse los que sean relevantes para la toma de decisiones.
- Compartir significa que una base de datos puede tener múltiples usos y usuarios. Una base de datos proporciona una memoria común para varias funciones en una organización. Por ejemplo, una base de datos de empleados puede servir para calcular la nómina, para hacer evaluaciones sobre desempeño, para hacer requerimientos de reportes del gobierno, etc. Muchos usuarios pueden acceder a la base de datos al mismo tiempo; por ejemplo, muchos clientes pueden hacer reservaciones en una aerolínea de forma simultánea. A menos que dos usuarios intenten modificar la misma parte de la base de datos al mismo tiempo, ambos pueden continuar sin tener que esperar al otro.
- Interrelación significa que los datos almacenados como unidades separadas se pueden conectar para mostrar un cuadro completo. Por ejemplo, una base de datos de clientes relaciona los datos de éstos (nombre, dirección, etc.) con los datos de una orden (número de orden, fecha de la orden, etc.) para facilitar su procesamiento. Las bases de datos contienen tanto entidades como relaciones entre entidades. Una entidad es un conjunto de datos generalmente sobre un tema, al que puede accederse de forma conjunta. Una entidad puede representar una persona, lugar, cosa o suceso; por ejemplo, una base de datos de empleados contiene entidades como empleados, departamentos y habilidades, así como las relaciones que muestran la asignación de los empleados con los departamentos, las habilidades que poseen y su historia salarial. Una base de datos típica de un negocio puede tener cientos de entidades y relaciones.

Para mostrar estas características, consideremos cierto número de bases de datos. Comencemos con una sencilla base de datos de una universidad (figura 1.1), ya que usted tiene cierta familiaridad con las actividades en una de ellas. La base de datos simplificada de una universidad

**FIGURA 1.1**  
Ilustración simplificada de la base de datos universitaria



Nota: Las palabras alrededor de la base de datos representan los procedimientos que ésta utiliza.

**FIGURA 1.2**  
Ilustración de la base de datos simplificada del departamento de distribución de agua



contiene información sobre los estudiantes, las facultades, los cursos, las ofertas de los cursos y las matriculaciones. La base de datos incluye procedimientos como registro de clases, asignación de profesores a los cursos ofertados, registro de calificaciones y calendarización de los cursos ofrecidos. Las relaciones en la base de datos de la universidad sirven para responder a preguntas como

- ¿Qué ofertas están disponibles para un curso en el periodo académico actual?
- ¿Quién es el instructor de un curso ofrecido?
- ¿Qué estudiantes están inscritos en un curso?

A continuación, consideremos una base de datos del departamento de distribución de agua, tal como se ilustra en la figura 1.2. La función principal del departamento de distribución de agua es facturar a los clientes por el uso de la misma. El consumo de agua de un cliente se mide de forma periódica y se prepara la factura. Existen muchos aspectos que influyen en la preparación de una factura, como el historial de pago del cliente, las características de la medición, el tipo de cliente (de bajos ingresos, arrendatario, propietario, pequeño negocio, gran empresa, etc.) y el ciclo de facturación. Las relaciones en la base de datos del departamento de distribución de agua sirven para contestar preguntas como

1. ¿En qué fecha se le envió la última factura al cliente?
2. ¿Cuánto consumo de agua se registró en la más reciente lectura del medidor de un cliente?
3. ¿Cuándo fue su último pago?

Finalmente, consideremos la base de datos de un hospital, como se ilustra en la figura 1.3. La base de datos del hospital apoya los tratamientos que los médicos realizan a los pacientes. Los médicos realizan diagnósticos y prescriben tratamientos con base en los síntomas. Diferentes proveedores de salud leen y contribuyen en el registro médico de un paciente. Las enfermeras

**FIGURA 1.3**  
**Ilustración de la base de datos simplificada de un hospital**



son responsables de monitorear los síntomas y administrar los medicamentos. El personal del comedor prepara las comidas de acuerdo con el plan de dietas. Los médicos prescriben nuevos tratamientos basándose en los resultados de tratamientos previos y de los síntomas de los pacientes. Las relaciones en la base de datos sirven para responder a preguntas como

- ¿Cuáles son los síntomas más recientes de un paciente?
- ¿Quién prescribió cierto tratamiento a un paciente?
- ¿Qué diagnóstico hizo el médico a un paciente?

Estas bases de datos simples carecen de muchos tipos de datos que se encuentran en las bases de datos reales. Por ejemplo, la base de datos simplificada de la universidad no contiene datos acerca de los requisitos de los cursos o de la capacidad de los salones de clases y sus ubicaciones. Las versiones reales de estas bases de datos deben contener muchas más entidades, relaciones y usos adicionales. Sin embargo, estas bases de datos simples tienen las características esenciales de las bases de datos empresariales: datos persistentes, múltiples usos y usuarios, y múltiples entidades conectadas por relaciones.

## 1.2 Características de los sistemas de administración de bases de datos

### sistema de administración de bases de datos (DBMS)

un conjunto de componentes que soportan la adquisición de datos, diseminación, mantenimiento, consultas y formateo.

Un sistema de administración de base de datos (DBMS, por sus siglas en inglés) es un conjunto de componentes que soportan la creación, el uso y el mantenimiento de bases de datos. Inicialmente, los DBMS proporcionaban un eficiente almacenamiento y recuperación de datos. Gracias a la demanda del mercado y a la innovación de productos, los DBMS han evolucionado para proporcionar un amplio rango de características para incorporar, almacenar, diseminar, mantener, recuperar y formatear datos. La evolución de estas funciones ha hecho que los DBMS sean más complejos. Puede tomar años de estudio y uso conocer por completo un DBMS en particular. Ya que los DBMS continúan evolucionando, usted debe actualizarse de forma continua en su conocimiento.

Para mostrar el alcance sobre las funciones que encontrará en los DBMS comerciales, la tabla 1.1 resume un conjunto común de funciones. El resto de esta sección presenta ejemplos de dichas funciones. Algunos ejemplos se extraen de Microsoft Access, un popular DBMS de escritorio. En capítulos posteriores se detallará más la introducción que aquí se presenta.

### 1.2.1 Definición de base de datos

**tabla**  
 se denomina a un arreglo de datos en dos dimensiones. Una tabla está formada por un encabezado y un cuerpo.

Para definir una base de datos deben especificarse las entidades y sus relaciones. En la mayoría de los DBMS comerciales, las tablas almacenan conjuntos de entidades. Una tabla (figura 1.4) tiene un renglón de encabezado (el primer renglón) que muestra los nombres de las columnas, y un cuerpo (los otros renglones) que muestra el contenido de la tabla. Las relaciones indican las conexiones entre tablas. Por ejemplo, la relación que conecta la tabla estudiante con la tabla inscripción muestra la oferta de cursos tomada por cada estudiante.

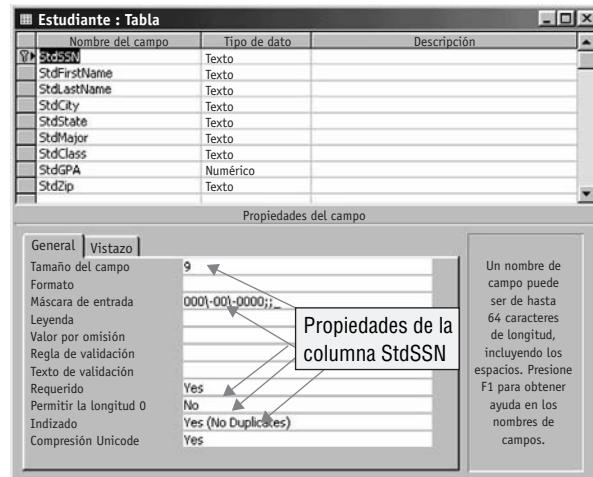
**TABLA 1.1**  
Resumen de las funciones generales de los DBMS

Función	Descripción
Definición de la base de datos	Lenguaje y herramientas gráficas para definir entidades, relaciones, restricciones de integridad y autorización de privilegios
Acceso no procedural	Lenguaje y herramientas gráficas para acceder a los datos sin necesidad de código complicado
Desarrollo de aplicaciones	Herramienta gráfica para desarrollar menús, formularios de captura de datos y reportes; los requerimientos de datos para los formularios y reportes se especifican utilizando un acceso no procedural
Interfase del lenguaje procedural	Lenguaje que combina el acceso no procedural con las capacidades totales de un lenguaje de programación
Procesamiento de transacciones	Mecanismos de control para prevenir la interferencia de usuarios simultáneos y recuperar datos perdidos en caso de una falla
Ajuste de la base de datos	Herramientas para monitorear y mejorar el desempeño de la base de datos

**FIGURA 1.4** Visualización de la Tabla Estudiante desde Microsoft Access

StdFirstName	StdLastName	StdCity	StdState	StdZip	StdMajor	StdClass	StdGPA
HOMER	WELLS	SEATTLE	WA	98121-1111	IS	FR	3.00
BOB	NORBERT	BOTHELL	WA	98011-2121	FIN	JR	2.70
CANDY	KENDALL	TACOMA	WA	99042-3321	ACCT	JR	3.50
WALLY	KENDALL	SEATTLE	WA	98123-1141	IS	SR	2.80
JOE	ESTRADA	SEATTLE	WA	98121-2333	FIN	SR	3.20
MARIAH	DODGE	SEATTLE	WA	98114-0021	IS	JR	3.60
TESS	DODGE	REDMOND	WA	98116-2344	ACCT	SO	3.30

**FIGURA 1.5**  
Ventana Definición de Tabla en Microsoft Access

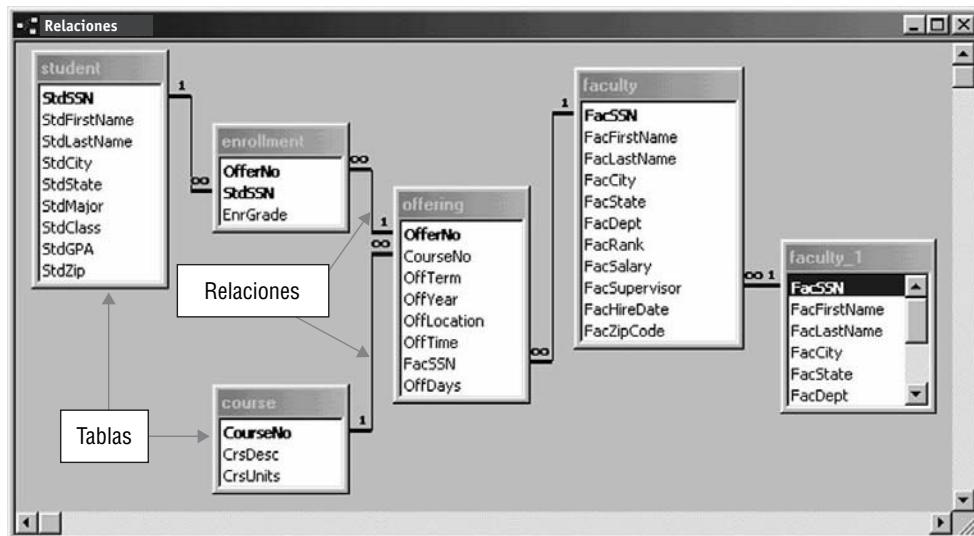


**SQL**  
lenguaje estándar de base de datos que incluye sentencias para la definición, la manipulación y el control de la base de datos.

La mayoría de los DBMS proporcionan diversas herramientas para definir bases de datos. El Lenguaje de Consulta Estructurada (SQL, por sus siglas en inglés) es un lenguaje estándar de la industria soportado por la mayoría de los DBMS. SQL se puede usar para definir tablas, relaciones entre ellas, restricciones de integridad (reglas que definen datos permitidos) y autorización de privilegios (reglas que restringen el acceso a los datos). El capítulo 3 describe las sentencias SQL para definir tablas y relaciones.

Además de SQL, muchos DBMS proporcionan herramientas gráficas orientadas a ventanas. Las figuras 1.5 y 1.6 ilustran herramientas gráficas para definir las tablas y sus relaciones. Si se usa la ventana Definición de Tabla de la figura 1.5, el usuario puede determinar las propiedades de las columnas, como los tipos de datos y el tamaño de los campos. Si se usa la ventana

**FIGURA 1.6**  
Ventana de definición  
Entidad Relación de  
Microsoft Access



de definición Entidad-Relación de la figura 1.6, se pueden definir las relaciones entre tablas. Una vez definida la estructura, la base de datos puede comenzar a poblararse. Los datos de la figura 1.4 deben agregarse cuando se han completado la ventana de Definición de Tablas y la ventana de Definición de Relaciones.

### 1.2.2 Acceso no procedural

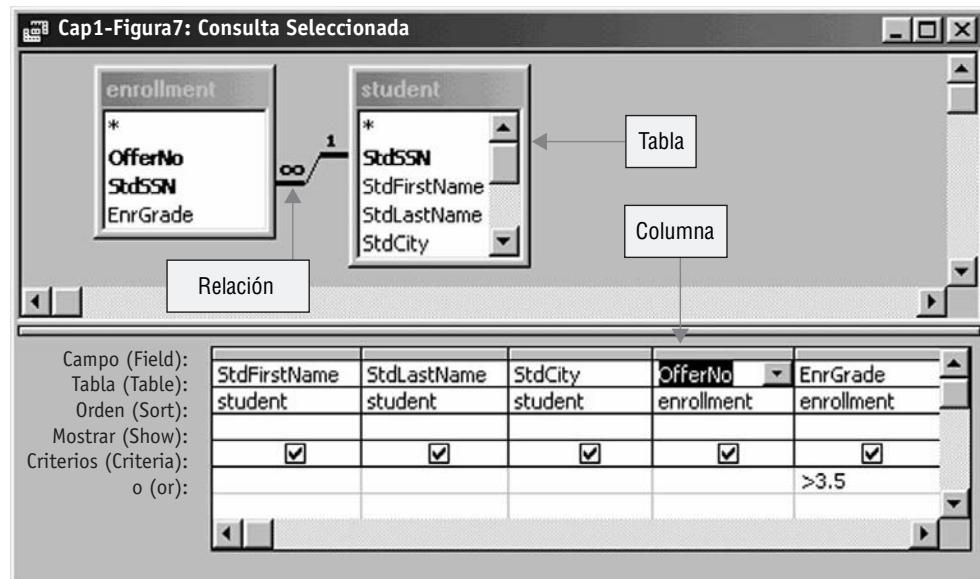
La función más importante de un DBMS es la habilidad de responder a las consultas. Una consulta es una solicitud de datos que responde a una pregunta. Por ejemplo, el usuario podría conocer a los clientes que tienen grandes estados de cuenta o productos con ventas fuertes en una región en particular. El acceso no procedural permite que los usuarios con habilidades computacionales limitadas realicen consultas. El usuario especifica las partes que desea extraer de una base de datos y no los detalles de implementación que ocurren con dicha extracción. Los detalles de la implementación involucran procedimientos de código complejo con bucles. Los lenguajes no procedurales no tienen sentencias de bucles (for, while, etc.), ya que sólo se especifican las partes a extraer de una base de datos.

Los accesos no procedurales pueden reducir el número de líneas de código por un factor de 100 cuando se comparan con un acceso procedural. Debido a que una gran parte del software de negocios involucra el acceso a datos, el acceso no procedural puede proporcionar una mejora dramática en la productividad del software. Para apreciar el significado del acceso no procedural consideremos su analogía con la planeación de unas vacaciones. Usted especifica el destino, el presupuesto de viaje, la duración y la fecha de salida. Estos hechos indican el “que” de su viaje. Para especificar el “cómo” de su viaje necesita indicar muchos detalles más, como la mejor ruta hacia su destino, su hotel preferido, la transportación terrestre, etc. El proceso de planeación es mucho más sencillo si tiene un profesional que le ayude con estos detalles adicionales. Como un profesional de la planeación, un DBMS lleva a cabo el proceso de planeación detallado para responder a las consultas expresadas en un lenguaje no procedural.

La mayoría de los DBMS proporcionan más de una herramienta para el acceso no procedural. La sentencia SELECT de SQL, descrita en el capítulo 4, constituye una forma no procedural para acceder a una base de datos. La mayoría de los DBMS también proporcionan herramientas gráficas para acceder a las bases de datos. La figura 1.7 ilustra una herramienta gráfica disponible en Microsoft Access. Para enviar una consulta a la base de datos el usuario únicamente tiene que indicar las tablas, las relaciones y las columnas requeridas. Access es responsable de generar el plan para obtener los datos solicitados. La figura 1.8 muestra el resultado de la ejecución de la consulta de la figura 1.7.

**lenguaje de base de datos no procedural**  
un lenguaje tal como SQL que permite especificar las partes a las cuales se tiene acceso en una base de datos sin tener que codificar un procedimiento complejo. Los lenguajes no procedurales no incluyen sentencias anidadas.

**FIGURA 1.7**  
Ventana de diseño de consultas de Microsoft Access



**FIGURA 1.8**  
Resultado de ejecutar la consulta de la figura 1.7

StdFirstName	StdLastName	StdCity	OfferNo	EnrGrade
MARIAH	DODGE	SEATTLE	1234	3.8
BOB	NORBERT	BOTHELL	5679	3.7
ROBERTO	MORALES	SEATTLE	5679	3.8
MARIAH	DODGE	SEATTLE	6666	3.6
LUKE	BRAZZI	SEATTLE	7777	3.7
WILLIAM	PILGRIM	BOTHELL	9876	4

### 1.2.3 Desarrollo de aplicaciones e interfase del lenguaje procedural

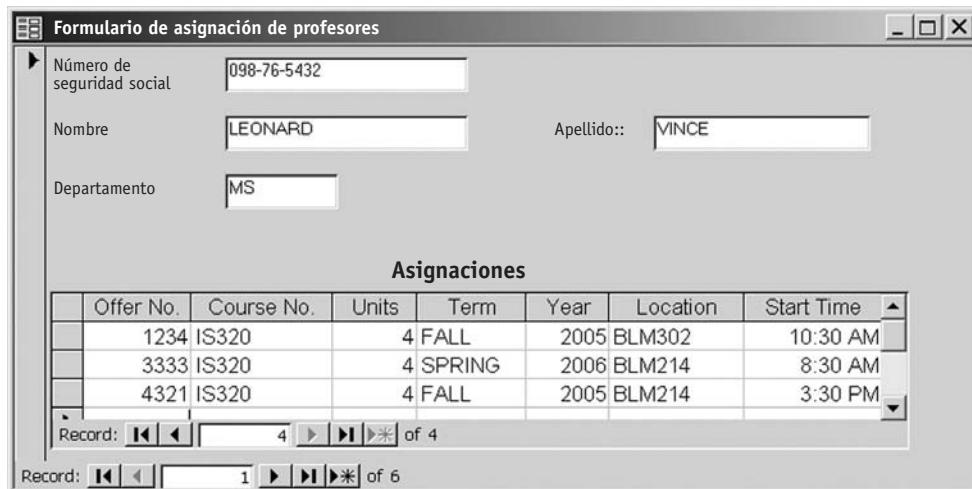
La mayoría de los DBMS van más allá del simple acceso a los datos, pues proporcionan herramientas gráficas para construir aplicaciones completas usando formularios y reportes. Los formularios para captura de datos proporcionan una herramienta adecuada para introducir y editar la información, mientras que los reportes mejoran la apariencia de los datos desplegados y su impresión. El formulario de la figura 1.9 puede usarse para agregar asignaciones de un curso nuevo para un profesor, o bien, para modificar las asignaciones existentes. El reporte de la figura 1.10 usa la sangría (la indentación) para mostrar los cursos impartidos en cada facultad y por departamento. El estilo de sangría puede ser más sencillo a la vista que el estilo tabular mostrado en la figura 1.8. Se pueden desarrollar muchas formas y reportes con una herramienta gráfica sin codificar de forma detallada. Por ejemplo, las figuras 1.9 y 1.10 se desarrollaron sin código. El capítulo 10 describe los conceptos implícitos en el desarrollo de formularios y reportes.

El acceso no procedural hace posible la creación de formularios y reportes sin un código extenso. Como parte de la creación de un formulario o reporte, el usuario indica los requerimientos de datos utilizando lenguaje carente de procedimientos (SQL) o una herramienta gráfica. Para completar la definición de un formulario o reporte, el usuario indica el formato de los datos, la interacción del usuario y todos los detalles.

Además de las herramientas para el desarrollo de aplicaciones, una interfase de lenguaje procedural agrega todas las capacidades de un lenguaje de programación. El acceso sin procedimientos y las herramientas de desarrollo de aplicaciones, aunque adecuadas y potentes, algunas veces no son lo suficientemente eficientes o no proporcionan el nivel de control necesario para el desarrollo de aplicaciones. Cuando estas herramientas no son adecuadas, los DBMS proporcionan todas las capacidades de un lenguaje de programación. Por ejemplo, Visual Basic for Applications (VBA) es un lenguaje de programación integrado a Microsoft

**interfase de lenguaje procedural**  
un método para combinar un lenguaje no procedural, como SQL, con un lenguaje de programación, como COBOL o Visual Basic.

**FIGURA 1.9**  
**Forma de Microsoft Access para asignar cursos a un profesor**



**FIGURA 1.10**  
**Reporte de Microsoft Access de la carga de trabajo de un profesor**

<b>Reporte de la carga de trabajo para el profesor para el año académico 2005-2006</b>							
Nombre del departamento	Periodo	Número ofrecido	Unidades	Límite	Inscritos	Porcentaje completado	Baja matrícula
FIN	JULIA MILLS						
	INVIERNO	5678	4	20	1	5.00%	<input checked="" type="checkbox"/>
	Resumen del 'periodo' = INVIERNO (1 registro detallado)						
	Suma		4		1		
	Promedio					5.00%	
	Resumen de JULIA MILLS						
	Suma		4		1		
	Promedio					5.00%	
	Resumen del 'departamento' = FIN (1 registro detallado)						

Access. VBA permite la personalización completa del acceso a la base de datos, procesamiento de formularios y generación de reportes. La mayoría de los DBMS comerciales tienen una interfase de lenguaje procedural comparable a VBA. Por ejemplo, Oracle tiene el lenguaje PL/SQL y Microsoft SQL Server tiene el lenguaje Transact-SQL. El capítulo 11 describe las interfaces del lenguaje procedural y del lenguaje PL/SQL.

#### 1.2.4 Funciones de soporte para las operaciones de base de datos

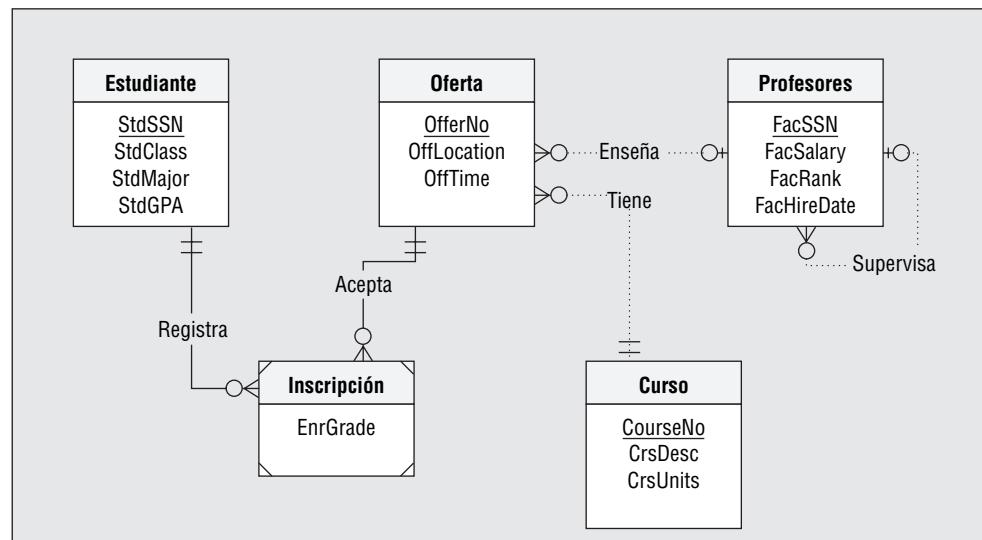
##### procesamiento de transacciones

procesamiento confiable y eficiente de un gran volumen de trabajo repetitivo. Los DBMS aseguran que los usuarios simultáneos no interfieran entre ellos y que las fallas no ocasionen la pérdida del trabajo.

El procesamiento de transacciones le permite a un DBMS procesar grandes volúmenes de trabajo repetitivo. Una transacción es una unidad de trabajo que se debe procesar de forma segura sin la interferencia de otros usuarios y sin perder los datos debido a fallas. Ejemplos de transacciones son el retiro de efectivo de un cajero automático, hacer una reservación en una aerolínea e inscribirse a un curso. Un DBMS se asegura de que las transacciones se encuentren libres de interferencia con otros usuarios, que las partes de una transacción no se pierdan cuando hay alguna falla, y que las transacciones no hagan que la base de datos se vuelva inconsistente. El procesamiento de transacciones es la mayoría de las veces un asunto “tras bambalinas”. El usuario no conoce los detalles acerca del procesamiento de las transacciones que no sea la seguridad en su realización.

La puesta a punto del rendimiento de una base de datos incluye cierto número de monitoreos y programas de utilerías que mejoran el desempeño. Algunos DBMS pueden monitorear el uso de una base de datos, la distribución de las múltiples partes de una base de datos y su crecimiento.

**FIGURA 1.11**  
Diagrama entidad-relación (ERD) para la base de datos universitaria



Los programas de utilerías se pueden proporcionar para reorganizar una base de datos, seleccionar las estructuras físicas para mejorar su desempeño y reparar sus partes dañadas.

El procesamiento de transacciones y las mejoras al rendimiento de una base de datos están presentes principalmente en los DBMS que soportan grandes bases de datos con muchos usuarios simultáneos. A estos DBMS se les conoce como corporativos, ya que las bases de datos que soportan, por lo general, son críticas para el funcionamiento de una organización. Los DBMS corporativos comúnmente se ejecutan en poderosos servidores de alto costo. En contraste, los DBMS de escritorio se ejecutan en computadoras personales y servidores pequeños que soportan un procesamiento limitado de transacciones, pero a un costo mucho menor. Los DBMS de escritorio soportan bases de datos utilizadas por equipos de trabajo y negocios pequeños. Los DBMS embebidos son una categoría emergente del software de bases de datos. Como su nombre lo especifica, un DBMS embebido reside en un sistema más grande, ya sea una aplicación o un dispositivo, tal como un asistente digital personal (PDA, por sus siglas en inglés) o una tarjeta inteligente. Los DBMS embebidos proporcionan un procesamiento transaccional limitado, pero tienen pocos requerimientos de memoria, procesamiento y almacenamiento.

### 1.2.5 Funciones de terceros

Además de las funciones proporcionadas de forma directa por los fabricantes de los DBMS, disponemos también de software de terceros para muchos de ellos. En la mayoría de los casos el software de terceros amplía las funciones disponibles en el software de bases de datos. Por ejemplo, muchos fabricantes ofrecen herramientas avanzadas de diseño de bases de datos que extienden sus capacidades de definición y rendimiento proporcionadas por un DBMS. La figura 1.11 muestra un diagrama de base de datos (un diagrama entidad-relación) creado con Visio Professional, una herramienta para el diseño de bases de datos. El ERD de la figura 1.11 se puede convertir en tablas soportadas por la mayoría de los DBMS comerciales. En algunos casos, el software de terceros compite de forma directa con el producto de la base de datos. Por ejemplo, los fabricantes proporcionan herramientas para el desarrollo de aplicaciones que pueden reemplazar a las provistas con la base de datos.

## 1.3 Desarrollo de la tecnología de base de datos y la estructura de mercado

La sección anterior mostró un breve recorrido por las funciones que se encuentran típicamente en un DBMS. Las funciones de los productos de hoy son una mejora significativa en relación con las de hace algunos años. La administración de base de datos, como muchas otras áreas de

**TABLA 1.2**  
**Breve evolución de la tecnología de base de datos**

Era	Generación	Orientación	Principales funciones
1960	Primera generación	Archivo	Estructuras de archivos e interfaces de programa propietario
1970	Segunda generación	Navegación en redes	Redes y jerarquías de registros relacionados, interfaces de programación estándar
1980	Tercera generación	Relacional	Lenguajes no procedurales, optimización, procesamiento transaccional
1990 a 2000	Cuarta generación	Objeto	Multimedia, activa, procesamiento distribuido, operadores más poderosos, procesamiento de data warehouse, habilitación para XML

la computación, ha tenido un tremendo crecimiento tecnológico. Con el fin de proporcionarle un contexto para apreciar los DBMS actuales, esta sección contempla los últimos cambios tecnológicos y sugiere las tendencias a futuro. Después de esta revisión, se presenta el mercado actual de las bases de datos.

### 1.3.1 Evolución de la tecnología de bases de datos

La tabla 1.2 muestra una breve historia de la tecnología de las bases de datos a través de cuatro generaciones de sistemas.<sup>1</sup> La primera generación soportaba búsquedas secuenciales y aleatorias, pero el usuario necesitaba escribir un programa de computadora para obtener el acceso. Por ejemplo, se puede escribir un programa para obtener todos los registros de clientes, o encontrar solamente el registro de un cliente con un número específico. Como los sistemas de primera generación no ofrecían suficiente soporte para relacionar datos, usualmente eran vistos como sistemas de procesamiento de archivos en vez de DBMS. Los sistemas de procesamiento de archivos pueden administrar sólo una entidad a la vez, a diferencia de las diversas entidades y relaciones de un DBMS.

Los productos de la segunda generación fueron los primeros DBMS reales, ya que podían administrar muchas entidades y relaciones. Sin embargo, para obtener el acceso a los datos todavía se tenía que escribir un programa. Los sistemas de segunda generación son conocidos como “navegacionales”, ya que el programador tenía que escribir código para navegar entre una red de registros ligados. Algunos de los productos de segunda generación se adhirieron a la definición estándar de una base de datos y del lenguaje de manipulación desarrollado por el Comité de Lenguajes de Sistemas de Datos (CODASYL, por sus siglas en inglés), una organización de estándares. El estándar CODASYL tuvo una limitada aceptación en el mercado gracias a que IBM, la compañía de cómputo dominante durante esta época, ignoró el estándar. IBM apoyó un enfoque distinto conocido como el modelo jerárquico de datos.

En lugar de enfocarse en el estándar de segunda generación, los laboratorios de investigación de IBM y las instituciones académicas desarrollaron las bases para una nueva generación de DBMS. El desarrollo más importante provino de los lenguajes no procedurales para el acceso a bases de datos. A los sistemas de la tercera generación se les conoce como DBMS relacionales, porque se basan en relaciones matemáticas y operadores asociados. La optimización de tecnología se desarrolló para que el acceso a través de lenguajes sin procedimientos fuera más eficiente. Como los lenguajes no procedurales proporcionaron una notoria mejoría sobre el acceso navegacional, los sistemas de la tercera generación reemplazaron a los de la segunda, y como su tecnología era tan diferente, muchos de los nuevos sistemas fueron desarrollados por nuevas compañías y no por los fabricantes de la generación previa. IBM fue la excepción más notoria: su peso ocasionó la adopción de SQL como un estándar ampliamente aceptado.

<sup>1</sup> Las generaciones de DBMS no deben confundirse con las generaciones de los lenguajes de programación. En particular, los lenguajes de cuarta generación se refieren a las características del lenguaje de programación, no a las características de los DBMS.

Los DBMS de cuarta generación están extendiendo las fronteras de la tecnología de bases de datos hacia datos no convencionales, la Internet y el proceso de los data warehouse. Los sistemas de cuarta generación pueden almacenar y manipular datos no convencionales como imágenes, videos, mapas, sonidos y animaciones. Debido a que estos sistemas consideran cada tipo de dato como un objeto a administrar, a los sistemas de cuarta generación algunas veces se les llama “orientados a objetos” o “relacionados con objetos”. El capítulo 18 presenta detalles acerca de las funciones de objetos de un DBMS. Además del énfasis que se pone en los objetos, en la Internet se publican DBMS para desarrollar nuevas formas de procesamiento distribuido. Ahora, la mayoría de los DBMS incluye formas adecuadas para publicar datos estáticos y dinámicos en Internet haciendo uso del Lenguaje de Marcaje Extendido (XML, por sus siglas en inglés) como un estándar. El capítulo 17 presenta los detalles acerca de las funcionalidades del procesamiento cliente-servidor dentro de los DBMS que soportan el acceso web a las bases de datos.

Un desarrollo reciente en los DBMS de cuarta generación es el soporte para el procesamiento de los data warehouse. Un data warehouse es una base de datos que apoya la toma de decisiones a mediano y largo plazos en las organizaciones. La extracción de datos resumidos domina el procesamiento del data warehouse, mientras que la combinación de actualizaciones y extracciones de datos ocurre en las bases de datos que soportan las operaciones diarias de una organización. El capítulo 16 presenta los detalles acerca de las funcionalidades de los DBMS que soportan el procesamiento de un data warehouse.

El mercado de los sistemas de cuarta generación es una batalla de fabricantes de sistemas de la tercera generación que están actualizando sus productos contra un nuevo grupo de desarrollo de sistemas de software de código abierto. Hasta ahora, parece que las compañías que ya existían llevan la delantera.

### 1.3.2 El mercado actual del software de base de datos

De acuerdo con la Sociedad Internacional de Datos (IDC), las ventas (licencias y mantenimiento) del software corporativo de bases de datos alcanzó los 13 600 millones de dólares en 2003, un incremento de 7.6 por ciento desde 2002. Los DBMS corporativos hacen uso de servidores *mainframe* que ejecutan el sistema operativo MVS de IBM, y de servidores de rango medio que ejecutan los sistemas operativos Unix (Linux, Solaris, AIX y otras variantes) y Microsoft Windows Server. Las ventas del software de bases de datos corporativo han seguido las condiciones económicas con grandes aumentos durante los años de gran impacto de Internet, seguidas por un crecimiento lento durante las pendientes de las punto com y las telecomunicaciones. Para el panorama a futuro, IDC estima que las ventas de los DBMS corporativos alcancen los 20 mil millones de dólares en 2008.

De acuerdo con IDC, tres son los productos que dominan el mercado del software de bases de datos corporativo, tal como se muestra en la tabla 1.3. La clasificación de IDC incluye tanto las ganancias por licencias como por mantenimiento. Si se consideran únicamente los costos de

**TABLA 1.3**  
**Distribución del mercado de software corporativo de bases de datos en 2003<sup>2</sup>**

Producto	Distribución total del mercado	Comentarios
Oracle 9i, 10g	39.9%	Domina el ambiente Unix; también con gran desempeño en el mercado Windows
IBM DB2, Informix	31.3%	Domina los ambientes MVS y AS/400; adquirió Informix en 2001; 25% de la distribución del mercado de los ambientes Unix
Microsoft SQL Server	12.1%	Domina el mercado de Windows; no tiene presencia en otros ambientes
Otros	16.7%	Incluye a Sybase, NCR Terradata, Progress Software, MySQL, PostgreSQL, Ingres de código abierto, Firebird y otros

<sup>2</sup> Distribución de mercado de acuerdo con un estudio de la Sociedad Internacional de Datos de 2004.

licencias, el Gartner Group coloca a IBM en la rebanada más grande del mercado, con 35.7 por ciento, seguido de Oracle con 33.4 por ciento y Microsoft con 17.7 por ciento. El mercado, en general, es muy competitivo entre las principales compañías y las pequeñas que agregan muchas funciones nuevas en cada versión.

Los productos DBMS de código abierto han comenzado a retar a los comerciales en la franja de mercado de DBMS de bajo rango. Aunque el código fuente de los productos DBMS de código abierto se encuentra disponible sin costo alguno, la mayoría de las organizaciones adquieren contratos de soporte, con lo que los productos de código fuente libre resultan no gratuitos. Aún así, muchas organizaciones han reportado un ahorro en los costos al usar productos DBMS de código fuente libre, en su mayoría para sistemas que no son de misión crítica. MySQL, que se introdujo por primera vez en 1995, es el líder del mercado de los DBMS de código fuente libre. PostgreSQL e Ingres de código abierto son productos maduros de DBMS de código fuente libre. Firebird es un nuevo producto de código abierto que está ganando terreno.

En el mercado de software personal de base de datos, Microsoft Access domina, debido en parte a la penetración de Microsoft Office. El software personal de base de datos se vende generalmente como parte del software de productividad de Office. Microsoft Office controla cerca del 90 por ciento del mercado de software de productividad, por lo que Access mantiene un dominio semejante dentro del mercado de bases de datos personales. Otros productos importantes dentro de las bases de datos personales son Paradox, Approach, FoxPro y FileMaker Pro.

Con el fin de proporcionar una cobertura tanto del software de bases de datos personal como del corporativo, este libro abarca tanto a Oracle como a Microsoft Access. Adicionalmente, el énfasis en el estándar SQL en las partes 2 y 5 proporciona una cobertura sobre el lenguaje de bases de datos para el resto de los productos.

Debido al crecimiento potencial de los dispositivos personales de cómputo, la mayoría de los fabricantes de DBMS se han dispuesto a entrar en el mercado de los DBMS embebidos. Este mercado actualmente está compartido por un pequeño grupo de compañías pequeñas como iAnywhere Solutions y Solid Information Technology, de forma conjunta con los fabricantes de DBMS corporativos Oracle e IBM.

## 1.4 Arquitecturas de los sistemas de administración de bases de datos

---

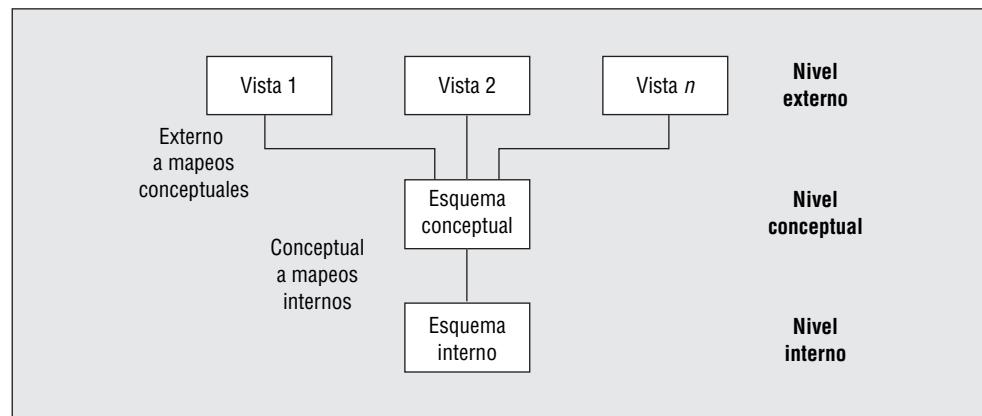
Para proporcionar un vistazo acerca de la organización interna de los DBMS, esta sección describe dos arquitecturas o marcos de trabajo organizacionales. La primera arquitectura describe una organización de definiciones de bases de datos para reducir el costo de mantenimiento del software. La segunda describe una organización de datos y software para apoyar el acceso remoto. Estas arquitecturas promueven un entendimiento conceptual en vez de indicar cómo es que se organiza un DBMS real.

### 1.4.1 Independencia de datos y la arquitectura de los tres esquemas

En los primeros DBMS existía una conexión cercana entre una base de datos y los programas que la utilizaban. En esencia, el DBMS era considerado como parte del lenguaje de programación. Como resultado, la definición de la base de datos formaba parte de los programas computacionales que accedían a la base de datos. Adicionalmente, el significado conceptual de base de datos no estaba separado de su implementación física en un disco magnético. Las definiciones acerca de la estructura de una base de datos y de su implementación física estaban mezcladas con los programas computacionales.

Esta asociación tan cercana entre las bases de datos y los programas relacionados condujo a problemas en el mantenimiento del software. Éste, incluyendo modificaciones, correcciones y mejoras, consumía una gran porción del presupuesto de desarrollo de software. En los primeros DBMS la mayoría de los cambios hechos en las definiciones de la base de datos ocasionaban modificaciones en los programas de cómputo. En muchos casos, los cambios en los programas computacionales involucraban la inspección detallada del código, un proceso

**FIGURA 1.12**  
Arquitectura de los tres esquemas



sumamente laborioso. Esta labor de inspección del código es semejante a la revisión hecha para el ajuste con el año 2000, donde los formatos de las fechas se modificaron a cuatro dígitos. Las mejoras del rendimiento de una base de datos eran difíciles, ya que en ocasiones cientos de programas computacionales necesitaban ser recompilados por cada modificación que se hacía. Debido a que eran comunes los cambios en la definición de las bases de datos, una gran parte de los recursos de mantenimiento del software se aplicaban para hacerlos. Algunos estudios han estimado un porcentaje tan alto como 50 por ciento de los recursos de mantenimiento de software.

El concepto de independencia de datos surgió para solucionar los problemas en el mantenimiento de programas. La independencia de datos significa que una base de datos debe tener una identidad separada de las aplicaciones que la usan (programas computacionales, formularios y reportes). La identidad separada permite que la definición de la base de datos cambie sin afectar las aplicaciones relacionadas. Por ejemplo, si se agrega una columna nueva a una tabla, las aplicaciones que no usen la columna nueva no deben verse afectadas. De forma similar, si se agrega una tabla nueva, sólo las aplicaciones que la requieran se verán afectadas. Esta separación debe marcarse aún más si algún cambio afecta únicamente la implementación física de la base de datos. Los especialistas en bases de datos deben poseer la libertad de experimentar mejoras al rendimiento sin tener que preocuparse por las modificaciones de los programas computacionales.

A mediados de la década de 1970, el concepto de independencia de datos condujo a la propuesta de la arquitectura de los tres esquemas, ilustrada en la figura 1.12. La palabra esquema, tal como se aplica a las bases de datos, significa descripción de la base de datos. La arquitectura de los tres esquemas incluye tres niveles de descripción de la base de datos. El nivel externo es el nivel del usuario. Cada grupo de usuarios puede tener una vista externa separada (o vista para abreviar) de una base de datos que se ajusta a las necesidades específicas del grupo.

En contraste, los esquemas conceptual e interno representan la base de datos por completo. El esquema conceptual define las entidades y las relaciones. Para una base de datos empresarial, el esquema conceptual puede ser muy grande, tal vez de cientos de tipos de entidades y relaciones. Tal como el esquema conceptual, el esquema interno representa la base de datos por completo; sin embargo, éste representa la vista de almacenamiento de la base de datos, mientras que el esquema conceptual representa el significado lógico de la base de datos. El esquema interno define archivos o conjuntos de datos en un dispositivo de almacenamiento como el disco duro. Un archivo puede almacenar una o más de las entidades descritas en el esquema conceptual.

Para simplificar aún más los tres niveles de esquemas, la tabla 1.4 muestra las diferencias entre las definiciones de la base de datos en los tres niveles de los esquemas utilizando ejemplos de las funciones descritas en la sección 1.2. Incluso, en la base de datos simplificada de una universidad las diferencias entre los niveles del esquema son claras. Con una base de datos más compleja las diferencias serán más marcadas, con muchas más vistas, un esquema conceptual más grande y uno interno más complejo.

Los mapeos de esquemas describen la manera en que un esquema de un nivel más alto se genera a partir de un esquema de nivel inferior. Por ejemplo, las vistas externas de la tabla 1.4

### independencia de datos

una base de datos debe tener una identidad separada de las aplicaciones que la usan (programas computacionales, formularios y reportes). La identidad separada permite que las definiciones de la base de datos se modifiquen sin afectar las aplicaciones relacionadas.

### arquitectura de los tres esquemas

una arquitectura para compartmentalizar las descripciones de la base de datos. La arquitectura de los tres esquemas se propuso como una forma de conseguir la independencia de datos.

**TABLA 1.4**

**Ejemplo de la base de datos de la universidad que ilustra las diferencias entre los niveles de esquema**

Nivel de esquema	Descripción
Externo	HighGPAView: datos solicitados para la consulta de la figura 1.7 FacultyAssignmentFormView: datos solicitados para el formulario de la figura 1.9 FacultyWorkLoadReportView: datos solicitados para el reporte de la figura 1.10
Conceptual	Tablas Estudiante, Inscripción, Curso, Profesor y sus relaciones (figura 1.6)
Interno	Archivos necesarios para almacenar las tablas; archivos adicionales para mejorar el desempeño (propiedad de indexación de la figura 1.5)

se generan a partir de las tablas del esquema conceptual. Los mapeos proporcionan el conocimiento para convertir una solicitud utilizando una vista externa (por ejemplo, HighGPAView) en una solicitud que usa las tablas del esquema conceptual. El mapeo entre los niveles conceptual e interno muestra cómo las entidades se almacenan en archivos.

Los DBMS utilizan esquemas y mapeos para asegurar la independencia de datos. Típicamente, las aplicaciones tienen acceso a una base de datos utilizando vistas. El DBMS convierte la solicitud de una aplicación en una solicitud, usando el esquema conceptual en lugar de la vista. Después, el DBMS transforma la solicitud del esquema conceptual en una solicitud usando el esquema interno. La mayoría de los cambios en el esquema conceptual o en el interno no afectan las aplicaciones, ya que no utilizan los niveles inferiores de los esquemas. El DBMS, y no el usuario, es responsable de usar los mapeos para hacer las transformaciones. Para conocer más detalles acerca de los mapeos y transformaciones refiérase al capítulo 10 que describe las vistas y las transformaciones entre los niveles conceptual y externo. El capítulo 8 describe la optimización de consultas, el proceso de convertir una consulta de nivel conceptual en una representación de nivel interno.

La arquitectura de los tres esquemas es un estándar oficial del Instituto Americano de Estándares Nacionales (ANSI); sin embargo, los detalles específicos del estándar nunca se adoptaron de forma global. En su lugar, el estándar sirve como una guía acerca de cómo se puede alcanzar la independencia de los datos. El espíritu de la arquitectura de los tres esquemas ha sido ampliamente implementado en los DBMS de tercera y cuarta generación.

#### 1.4.2 Procesamiento distribuido y la arquitectura cliente-servidor

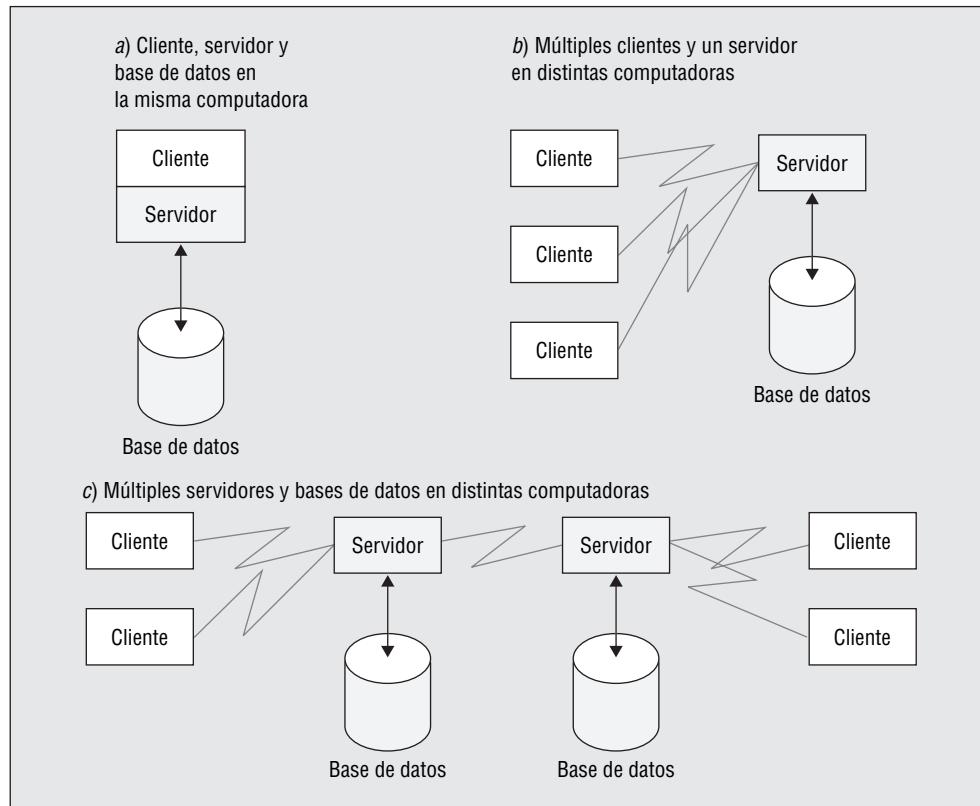
Con la creciente importancia de las redes de cómputo y la Internet, el procesamiento distribuido se está convirtiendo en una función vital de los DBMS. El procesamiento distribuido permite que computadoras que se encuentran dispersas geográficamente cooperen para proporcionar el acceso a los datos. Una gran parte del comercio electrónico de Internet involucra el acceso y la actualización de bases de datos remotas. Muchas bases de datos de las tiendas departamentales, bancos y comercio seguro se encuentran disponibles en Internet. Los DBMS utilizan la capacidad disponible de la red y del procesamiento local para proporcionar un acceso remoto eficiente.

Muchos DBMS soportan el procesamiento distribuido haciendo uso de una arquitectura cliente-servidor. Un cliente es un programa que envía solicitudes a un servidor. Un servidor procesa las solicitudes del cliente. Por ejemplo, un cliente puede solicitar a un servidor que extraiga los datos de un producto. Éste localiza los datos y los envía de vuelta al cliente, quien puede realizar algún procesamiento adicional a los datos antes de desplegar los resultados al usuario. Otro ejemplo: un cliente envía una orden completa a un servidor. El servidor valida la orden, actualiza la base de datos y envía un acuse de recibo al usuario. El cliente informa al usuario que la orden ha sido procesada.

Para mejorar el desempeño y la disponibilidad de los datos, la arquitectura cliente-servidor soporta muchas formas de distribuir el software y los datos en una red de cómputo. El esquema más simple consiste en colocar tanto el software como los datos en una misma computadora (figura 1.13a). El software y los datos también pueden distribuirse para aprovechar las ventajas de una red. En la figura 1.13b), el software del servidor y la base de datos se ubican en una computadora remota. En la figura 1.13c), el software del servidor y la base de datos se ubican en varias computadoras remotas.

**arquitectura cliente-servidor**  
un arreglo de componentes (clientes y servidores) y datos entre computadoras conectadas a una red. La arquitectura cliente-servidor soporta el procesamiento eficiente de mensajes (solicitudes de servicio) entre los clientes y los servidores.

**FIGURA 1.13**  
Arreglos típicos de cliente-servidor entre una base de datos y el software



El DBMS tiene varias responsabilidades en una arquitectura cliente-servidor; proporciona el software que se puede ejecutar en el cliente y en el servidor. Es común que el software del cliente sea responsable de aceptar los comandos del usuario, desplegar los resultados y realizar algún procesamiento a los datos. El software del servidor valida las peticiones de los clientes; localiza las bases de datos remotas; las actualiza, en caso de ser necesario, y envía los datos al cliente en un formato que éste pueda entender.

Las arquitecturas cliente-servidor proporcionan una forma flexible para que los DBMS interactúen con las computadoras de la red. La distribución del trabajo entre clientes y servidores y las alternativas disponibles para ubicar los datos y el software son mucho más complejas que las que aquí se muestran. Usted conocerá más detalles acerca de las arquitecturas cliente-servidor en el capítulo 17.

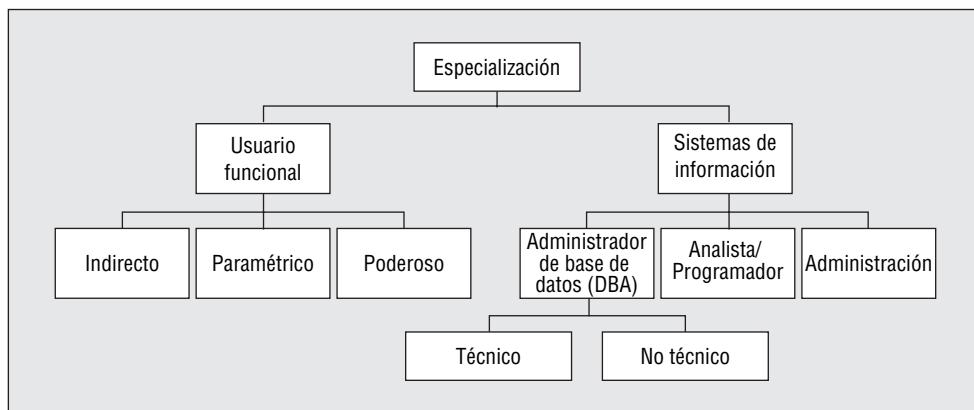
## 1.5 Impactos organizacionales de la tecnología de base de datos

Esta sección complementa la introducción a la tecnología de bases de datos planteando los efectos que tiene sobre las organizaciones. La primera sección describe las posibles interacciones que usted puede tener con alguna base de datos en una organización. La segunda describe la administración de los recursos de información en un esfuerzo por controlar los datos producidos y utilizados por una organización. Hemos puesto especial atención a los roles administrativos que usted puede jugar como parte del esfuerzo por controlar los recursos de información. El capítulo 14 proporciona más detalles acerca de las herramientas y los procesos utilizados por estos roles administrativos.

### 1.5.1 Interactuando con las bases de datos

Debido a la amplia presencia de las bases de datos, existen distintas formas en las que se puede interactuar con ellas. La clasificación de la figura 1.14 muestra las diferencias entre los usuarios funcionales, quienes interactúan con las bases de datos como parte de su trabajo, y

**FIGURA 1.14**  
Clasificación de roles



**TABLA 1.5**  
Responsabilidades  
del administrador  
de base de datos

Técnicas	No técnicas
Diseño de esquemas conceptuales	Implementar los estándares de la base de datos
Diseño de esquemas internos	Crear los materiales de entrenamiento
Monitoreo del desempeño de la base de datos	Promover los beneficios de las bases de datos
Seleccionar y evaluar el software de base de datos	Atender consultas de los usuarios
Diseñar las bases de datos cliente-servidor	
Solucionar los problemas de la base de datos	

los profesionales de los sistemas de información quienes participan en su diseño e implementación. Cada elemento de la jerarquía representa un rol que se puede jugar; aunque puede jugarse más de uno de forma simultánea. Por ejemplo, un usuario funcional con un trabajo como analista financiero puede jugar los tres roles en distintas bases de datos. En algunas organizaciones se traslapa la diferencia entre usuarios funcionales y profesionales de la información. En estas organizaciones, los usuarios funcionales pueden participar en el diseño y uso de las bases de datos.

Los usuarios funcionales pueden desempeñar un papel pasivo o activo cuando interactúan con las bases de datos. El uso indirecto de una base de datos es un rol pasivo. A un usuario indirecto se le ofrece un reporte de algunos datos extraídos de una base de datos. Un usuario paramétrico es más activo que un usuario indirecto. Un usuario paramétrico solicita formularios existentes o reportes con el uso de parámetros, valores de entrada que cambian de un uso a otro. Por ejemplo, un parámetro puede indicar un rango de fechas, un territorio de ventas o el nombre de un departamento. El usuario experimentado es el más activo. Debido a que la necesidad de toma de decisiones no se puede predecir, es importante dominar el uso no planeado de una base de datos. Un usuario experimentado es lo suficientemente hábil como para construir un formulario o un reporte cuando lo necesite. Los usuarios con experiencia deben comprender perfectamente los accesos no procedurales, una habilidad descrita en las partes 2 y 5 de este libro.

Los profesionales en sistemas de información interactúan con las bases de datos como parte del desarrollo de un sistema de información. Los analistas/programadores son responsables de obtener los requerimientos, diseñar las aplicaciones e implementar los sistemas de información. Ellos crean y usan vistas externas para desarrollar formularios, reportes y otras partes del sistema de información. Los directivos tienen el papel de supervisión en el desarrollo de las bases de datos y de los sistemas de información.

Los administradores de bases de datos ayudan tanto a los profesionales en sistemas de información como a los usuarios funcionales. Los administradores de bases de datos tienen una amplia variedad de responsabilidades técnicas y no técnicas (tabla 1.5). Las habilidades técnicas están más orientadas a los detalles; las no técnicas están más orientadas a las personas. La principal responsabilidad técnica es diseñar la base de datos; del lado no técnico, el tiempo del administrador de base de datos se divide entre varias actividades. Los administradores de bases de datos también pueden tener responsabilidades en la planeación y en la evaluación de DBMS.

**administrador de base de datos**  
una posición de apoyo que se especializa en la administración de bases de datos individuales y DBMS.

### 1.5.2 Administración de recursos de información

La administración de recursos de información es una respuesta al reto de utilizar de forma efectiva la tecnología de la información. El objetivo de la administración de recursos de información es utilizar la tecnología de la información como una herramienta de procesamiento, distribución e integración a lo largo de una organización. La administración de recursos de información tiene muchas semejanzas con la administración de recursos físicos, como el inventario. La administración de inventarios implica actividades como la salvaguarda del inventario para evitar su robo o deterioro, el almacenamiento para un uso eficiente, la selección de proveedores, el manejo de mermas, la coordinación del movimiento y la reducción de costos. La administración de recursos de información incluye actividades similares: planeación y adquisición de datos, protección de los datos de accesos no autorizados, aseguramiento de la confiabilidad, coordinación del flujo entre sistemas de información y eliminación de duplicados.

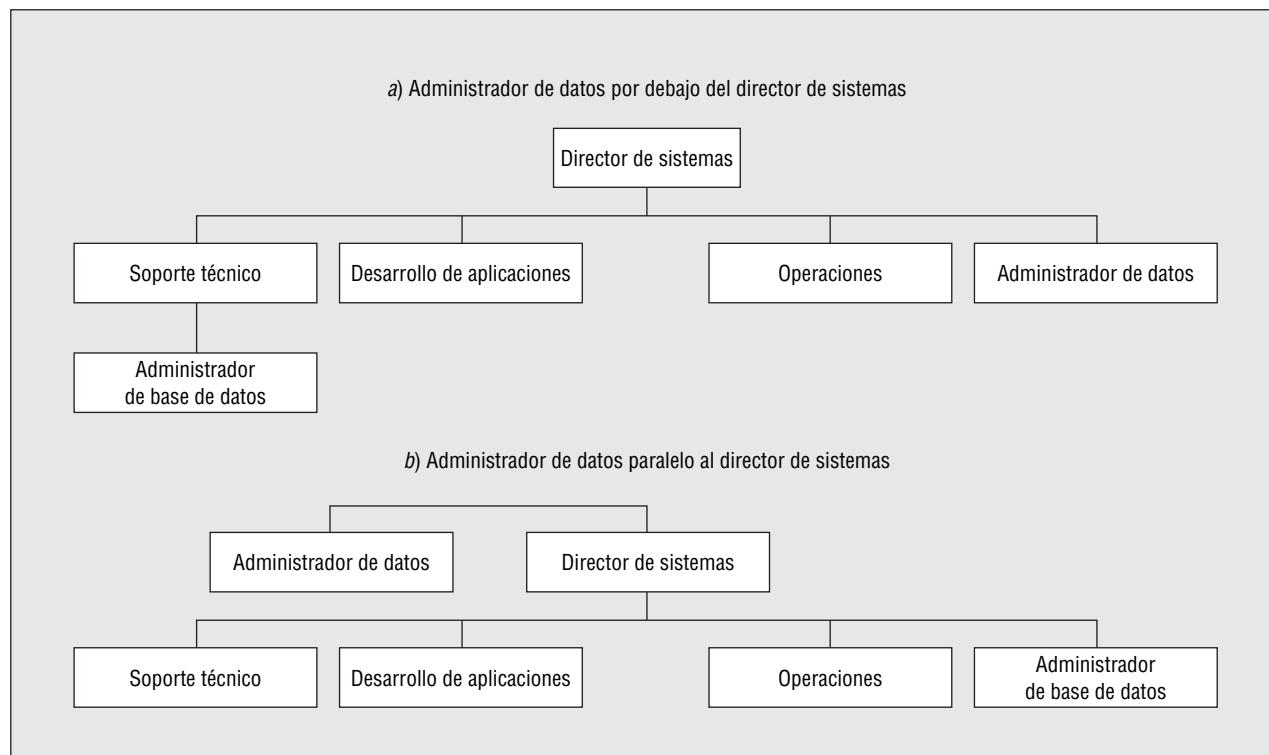
#### **administrador de datos**

una posición gerencial que realiza la planeación y establece las políticas para los recursos de información de una organización.

Como parte del control de recursos de información han surgido nuevas responsabilidades administrativas. El administrador de datos es una posición gerencial con muchas de estas responsabilidades, siendo la principal, la planeación del desarrollo de nuevas bases de datos. El administrador de datos conserva una arquitectura corporativa que describe las bases de datos existentes y las nuevas; también evalúa las nuevas tecnologías de la información y define los estándares para administrar las bases de datos.

El administrador de datos, generalmente, tiene responsabilidades más amplias que el administrador de base de datos. El papel principal del primero está en la planeación, mientras que el segundo tiene un rol más técnico, enfocado a bases de datos individuales y DBMS. El administrador de datos también revisa las fuentes de información desde un contexto más amplio y considera todo tipo de datos, digitales y no digitales. Muchas organizaciones están haciendo un gran esfuerzo para digitalizar los datos no tradicionales como videos, materiales de entrenamiento, imágenes y correspondencia. El administrador de datos desarrolla planes a largo plazo para los datos no tradicionales, mientras que el administrador de base de datos implementa estos planes usando la tecnología de base de datos apropiada.

**FIGURA 1.15** Ubicación en el organigrama de los administradores de datos y de bases de datos



Dado el amplio rango de responsabilidades, el administrador de datos normalmente tiene una posición superior en el organigrama. La figura 1.15 ilustra dos posibles ubicaciones de los administradores de datos y los administradores de bases de datos. En una organización pequeña se pueden combinar ambos roles en la administración de sistemas.

## Reflexión final

El capítulo 1 le ha ofrecido una amplia introducción a los DBMS. Debe utilizar estos antecedentes como un contexto para las habilidades que adquirirá en los siguientes capítulos. Aprendió que las bases de datos contienen datos interrelacionados que se pueden compartir en las diversas áreas de una organización. Los DBMS soportan la transformación de los datos para la toma de decisiones. Para soportar esta transformación, la tecnología de bases de datos ha evolucionado desde el simple acceso de archivos hasta poderosos sistemas que soportan la definición de bases de datos; accesos sin procedimientos; desarrollo de aplicaciones; procesamiento de transacciones, y mejoras del rendimiento. El acceso no procedural es el elemento más importante, porque permite el acceso sin un código detallado. Aprendió acerca de las dos arquitecturas que proveen los principios de organización para los DBMS. La arquitectura de los tres esquemas sustenta la independencia de datos, un concepto importante para reducir los costos del mantenimiento de software. Las arquitecturas cliente-servidor permiten que las bases de datos estén disponibles desde redes de computadoras, una función vital en el mundo interconectado de hoy en día.

Las habilidades desarrolladas en capítulos posteriores le permitirán trabajar como un usuario funcional activo o analista. Ambos tipos de usuarios necesitan comprender las habilidades que se describen en la segunda parte del libro. La parte 5 del libro proporciona habilidades para los analistas/programadores. Este libro también proporciona las habilidades fundamentales para obtener una posición de especialista como administrador de datos o de bases de datos. Las habilidades de la tercera, cuarta, sexta y séptima partes de este libro son muy útiles para la posición de administrador de bases de datos. Sin embargo, es probable que necesite tomar cursos adicionales, aprender detalles sobre los DBMS más populares y adquirir experiencia administrativa, antes de obtener un puesto de especialista. Una posición de especialista en bases de datos puede ser una oportunidad emocionante y una carrera lucrativa digna de tomarse en cuenta.

## Revisión de conceptos

- Características de una base de datos: persistente, interrelacionada y compartida.
- Funciones de los sistemas de administración de bases de datos (DBMS).
- Acceso no procedural: una clave en la productividad del software.
- Transacción: una unidad de trabajo que se debe procesar de forma confiable.
- Desarrollo de aplicaciones usando accesos sin procedimientos para especificar los requerimientos de datos de los formularios y los reportes.
- Interfase de lenguaje procedural para combinar el acceso no procedural con un lenguaje de programación como COBOL o Visual Basic.
- Evolución del software de bases de datos a lo largo de cuatro generaciones de mejoras tecnológicas.
- Enfoque actual en software de bases de datos que soporta multimedia, procesamiento distribuido, operadores más poderosos y data warehouse.
- Tipos de DBMS: corporativos, de escritorio, embebidos.
- Independencia de datos para solucionar los problemas del mantenimiento de programas de cómputo.
- Arquitectura de tres esquemas para reducir el impacto de los cambios en la definición de bases de datos.

- Arquitectura cliente-servidor para usar bases de datos sobre redes de cómputo.
- Roles de especialistas en bases de datos: administrador de bases de datos y administrador de datos.
- Administración de recursos de la información para el aprovechamiento de la tecnología de la información.

## Preguntas

1. Mencione una base de datos que haya utilizado en algún trabajo o como consumidor. Enliste las entidades y relaciones que contenga la base de datos. Si no está seguro, imagine qué entidades y relaciones tiene la base de datos.
2. Para la base de datos de la pregunta 1, enliste los grupos de usuarios que utilizan la base de datos.
3. Para uno de los grupos de la pregunta 2, describa una aplicación (formulario o reporte) que use el grupo.
4. Explique la propiedad de persistencia de las bases de datos.
5. Explique la propiedad de interrelación de las bases de datos.
6. Explique la propiedad de compartir de las bases de datos.
7. ¿Qué es un DBMS?
8. ¿Qué es SQL?
9. Mencione la diferencia entre un lenguaje procedural y uno no procedural. ¿Qué sentencias pertenecen a un lenguaje procedural, pero no a uno no procedural?
10. ¿Por qué el acceso no procedural es una función importante de los DBMS?
11. ¿Cuál es la conexión entre el acceso no procedural y el desarrollo de la aplicación (formulario o reporte)? ¿Puede el acceso no procedural utilizarse en el desarrollo de aplicaciones?
12. ¿Cuál es la diferencia entre un formulario y un reporte?
13. ¿Qué es la interfase de un lenguaje procedural?
14. ¿Qué es una transacción?
15. ¿Qué funciones proporciona un DBMS para apoyar el procesamiento transaccional?
16. Para la base de datos de la pregunta 1, mencione una transacción que sea utilizada. ¿Qué tan seguido piensa que la transacción se envía a la base de datos? ¿Cuántos usuarios envían transacciones al mismo tiempo? Adivine las últimas dos partes si no está seguro.
17. ¿Qué es un DBMS corporativo?
18. ¿Qué es un DBMS de escritorio?
19. ¿Qué es un DBMS embebido?
20. ¿Cuáles eran las funciones principales de los DBMS de primera generación?
21. ¿Cuáles eran las funciones principales de los DBMS de segunda generación?
22. ¿Cuáles eran las funciones principales de los DBMS de tercera generación?
23. ¿Cuáles son las funciones principales de los DBMS de cuarta generación?
24. Para la base de datos descrita en la pregunta 1, trace una tabla que ilustre las diferencias entre los niveles del esquema. Use la tabla 1.4 como guía.
25. ¿Cuál es el propósito de los mapeos en la arquitectura de los tres esquemas? ¿Es el usuario o el DBMS el responsable de usar los mapeos?
26. Explique cómo es que la arquitectura de los tres esquemas soporta la independencia de datos.
27. En una arquitectura cliente-servidor, ¿por qué las capacidades de procesamiento se dividen entre un cliente y un servidor? En otras palabras, ¿por qué el servidor no hace todo el procesamiento?
28. En una arquitectura cliente-servidor, ¿por qué, en ocasiones, se almacenan los datos en varias computadoras en lugar de en una sola?
29. Para la base de datos de la pregunta 1, mencione cómo interactúan los usuarios funcionales con la base de datos. Intente identificar a los usuarios indirectos, paramétricos y poderosos de la base de datos.
30. Explique las distintas responsabilidades de un usuario funcional activo y un analista. ¿Qué nivel de esquema utilizan ambos usuarios?
31. ¿Qué puesto le gustaría más como un objetivo a largo plazo en su carrera: administrador de bases de datos o administrador de datos? Explique brevemente su preferencia.
32. ¿Qué nicho de mercado ocupan los productos DBMS de código libre?

## Problemas

Dada la naturaleza introductoria de este capítulo, no existen problemas en él. Los problemas aparecen al final de la mayoría de los capítulos en el resto del libro.

## Referencias para ampliar su estudio

Los sitios web de la *DBAZine* ([www.dbazine.com](http://www.dbazine.com)), de la revista *Intelligent Enterprise* ([www.iemagazine.com](http://www.iemagazine.com)) y del *Advisor:com* ([www.advisor.com](http://www.advisor.com)) proporcionan información técnica y detallada acerca de DBMS comerciales, diseño de bases de datos y desarrollo de aplicaciones de bases de datos. Para aprender más acerca del rol de los especialistas de bases de datos y de la administración de los recursos de información, debe consultar Mullin (2002).

# Capítulo 2

# Introducción al desarrollo de bases de datos

## Objetivos de aprendizaje

Este capítulo constituye un repaso al proceso de desarrollo de bases de datos. Al finalizar este capítulo habrá adquirido los siguientes conocimientos y habilidades:

- Listar los elementos del ciclo de vida de un sistema de información.
- Describir el papel de una base de datos en un sistema de información.
- Explicar los objetivos del desarrollo de una base de datos.
- Comprender las relaciones entre las fases del proceso de desarrollo de una base de datos.
- Nombrar las características comunes que ofrecen las herramientas CASE para el desarrollo de bases de datos.

## Panorama general

El capítulo 1 nos ofreció una amplia introducción al uso de bases de datos dentro de las organizaciones, así como de la tecnología de bases de datos. Asimismo, se abordaron las características de una base de datos de negocios, las características básicas de un sistema de administración de bases de datos (DBMS), la arquitectura para el desarrollo de bases de datos y la interacción de roles organizacionales con las bases de datos. Este capítulo continúa con la introducción a la administración de bases de datos ofreciendo un amplio enfoque en el desarrollo de las bases de datos. Aprenderá sobre el contexto, objetivos, fases y herramientas de desarrollo de una base de datos para obtener las habilidades y conocimientos necesarios para abordar las partes 3 y 4.

Antes de desarrollar estas habilidades específicas se necesita comprender el amplio contexto del desarrollo de las bases de datos. Este capítulo explica el contexto de las bases de datos consideradas como parte de un sistema de información. Con ello usted aprenderá sobre los componentes de un sistema de información, su ciclo de vida y el papel del desarrollo de bases de datos como parte del desarrollo de los sistemas de información. El contexto de los sistemas de información representa los antecedentes para el desarrollo de bases de datos. De esta forma aprenderá, pues, las fases, habilidades y herramientas de software para el desarrollo de bases de datos.

## 2.1 Sistemas de información

Las bases de datos son parte de un sistema de información. Antes de comprender su desarrollo, debe comprender el contexto que lo rodea. Esta sección describe los componentes de un sistema de información y varias metodologías para su desarrollo.

### 2.1.1 Componentes de los sistemas de información

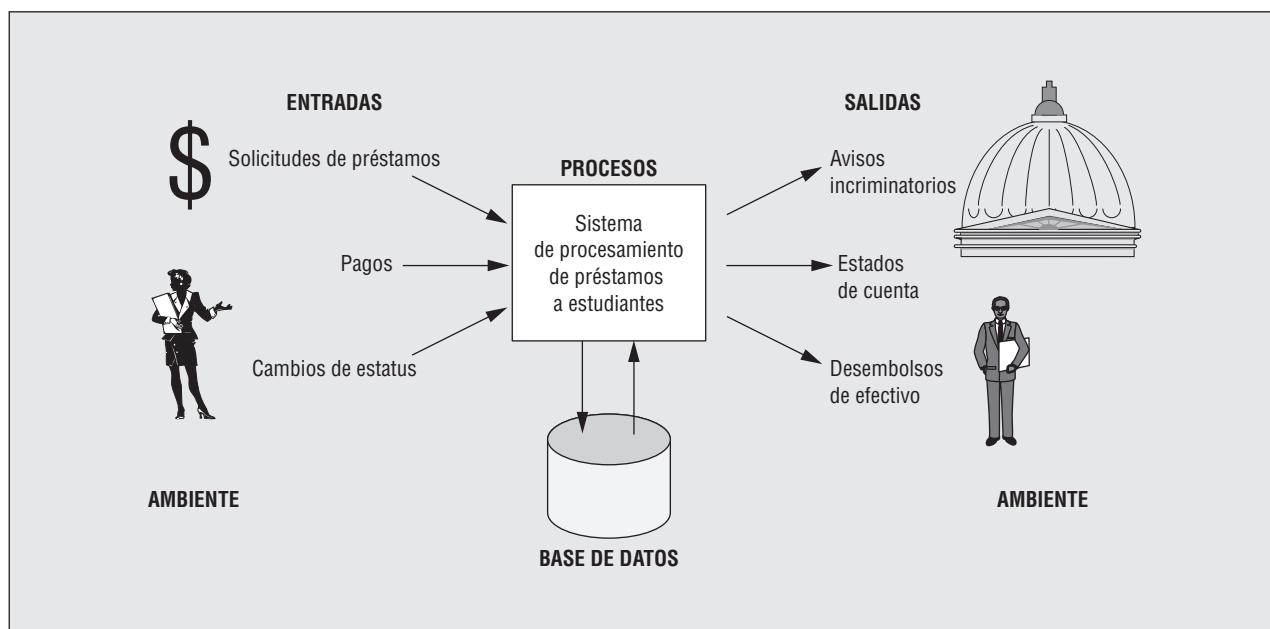
Un sistema es un conjunto de componentes relacionados entre sí que trabajan para alcanzar un objetivo. Los objetivos se cumplen mediante la interacción con el ambiente y funciones de desempeño. Por ejemplo, el sistema circulatorio de los seres humanos, formado por la sangre, las venas y el corazón, hace que la sangre circule hacia las distintas partes del cuerpo. El sistema circulatorio interactúa con otros sistemas del cuerpo para asegurar que la cantidad y la composición correcta de la sangre llegue a tiempo a distintas partes del cuerpo.

Un sistema de información es similar a un sistema físico (tal como el sistema circulatorio), con la excepción de que un sistema de información maneja datos en lugar de objetos físicos, como la sangre. Un sistema de información acepta datos provenientes de su entorno, los procesa y genera datos de salida para la toma de decisiones. Por ejemplo, un sistema de información para procesar los préstamos de dinero a los estudiantes (figura 2.1) ayuda al proveedor del servicio a rastrear los préstamos en las distintas instituciones a las que se ofrece. El entorno de este sistema está formado por los prestamistas, estudiantes y agencias de gobierno. Los prestamistas envían las solicitudes de préstamo aprobadas y los estudiantes reciben el efectivo para sus gastos escolares. Después de la graduación, los estudiantes reciben un estado de cuenta mensual y realizan los pagos para liquidar sus préstamos. Si un estudiante falla, la agencia gubernamental recibe un aviso incriminatorio.

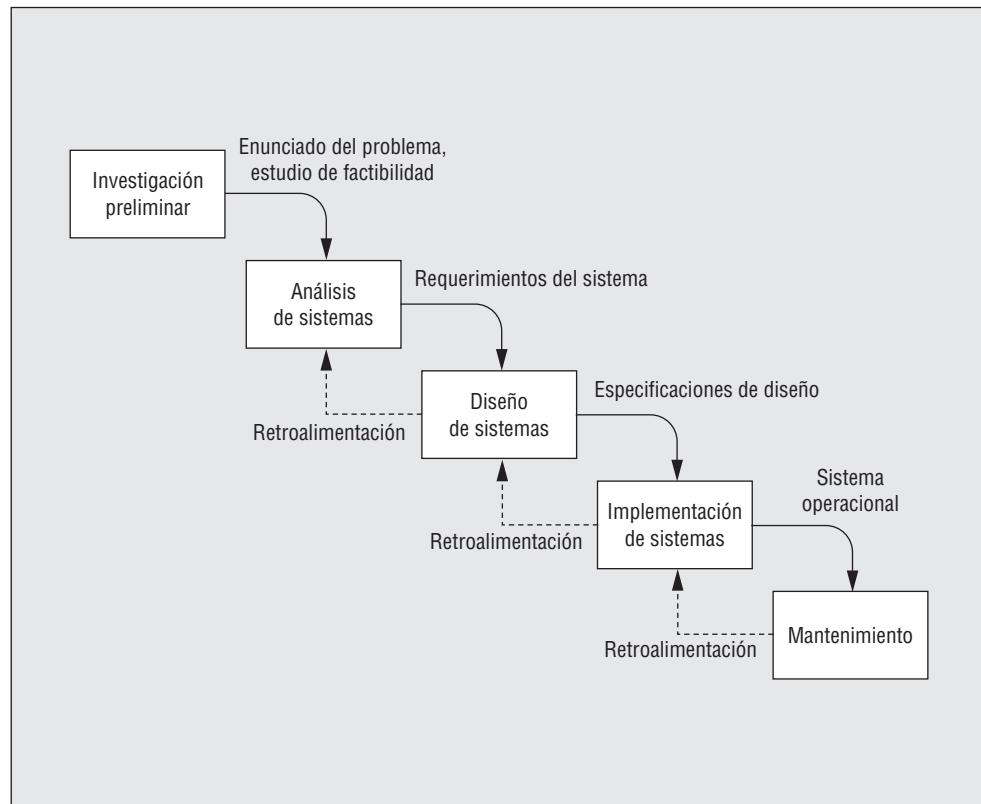
Las bases de datos son componentes esenciales de muchos sistemas de información. El papel de una base de datos es proporcionar una memoria de largo plazo para un sistema de información. La memoria de largo plazo contiene entidades y relaciones. Por ejemplo, la base de datos de la figura 2.1 contiene datos acerca de los estudiantes, préstamos y pagos, para poder generar estados de cuenta, desembolsos de efectivo y avisos incriminadores. Los sistemas de información que carecen de una memoria permanente, o que sólo tienen algunas de las variables de una memoria permanente, generalmente se incluyen en dispositivos que proporcionan un rango limitado de funciones, en vez del amplio rango de funciones que ofrece un sistema de información.

Las bases de datos no son los únicos componentes de los sistemas de información; también contienen personas, procedimientos, datos de entrada, datos de salida, software y hardware. Por lo tanto, el desarrollo de un sistema de información involucra más que el simple desarrollo de una base de datos, tal como lo comentaremos enseguida.

**FIGURA 2.1** Panorama del sistema de procesamiento de préstamos a estudiantes



**FIGURA 2.2**  
Ciclo de vida tradicional del desarrollo de sistemas



### 2.1.2 Proceso del desarrollo de sistemas de información

La figura 2.2 muestra las fases del ciclo de vida de un desarrollo de sistemas tradicional. Las fases particulares del ciclo de vida no son estándar. Distintos autores y organizaciones han propuesto de tres a 20 fases. Al ciclo de vida tradicional comúnmente se le conoce como modelo o metodología de cascada, ya que el resultado de cada una de sus fases fluye a la siguiente. El ciclo de vida tradicional generalmente es un marco de referencia. Para la mayoría de los sistemas, la frontera entre una fase y otra se disipa por lo que hay un considerable trabajo de retroceso entre ellas, para continuar nuevamente con la fase en que estábamos (*backtracking*). Pero el ciclo de vida tradicional aún es útil, ya que describe el tipo de actividad y muestra los detalles adicionales hasta que el sistema operacional emerge. Los siguientes elementos describen las actividades de cada fase:

- **Fase de investigación preliminar:** Genera el enunciado del problema y un estudio de factibilidad. El estudio de factibilidad identifica los costos y los beneficios del sistema. Si el sistema es factible, se otorga la aprobación para comenzar su análisis.
- **Fase de análisis de sistemas:** Genera los requerimientos que describen los procesos, datos e interacciones con el entorno. Se usan técnicas de diagramación para documentar los procesos, datos e interacciones en el entorno. Para generar los requerimientos, se estudia el sistema actual y se entrevista a los usuarios del sistema propuesto.
- **Fase de diseño de sistemas:** Genera un plan para implementar los requerimientos de forma eficiente. Se crean las especificaciones de diseño de los procesos, datos e interacciones con el entorno. Las especificaciones de diseño se enfocan a las alternativas que optimicen los recursos dadas las restricciones que se presenten.
- **Fase de implementación de sistemas:** Genera el código ejecutable, bases de datos y documentación para el usuario. Para implementar el sistema se codifican y prueban las especificaciones del diseño.

Antes de que el nuevo sistema sea operacional, debe crear un plan de transición del sistema existente al nuevo. Para obtener total confianza y experiencia con el nuevo sistema, una empresa debe utilizar ambos sistemas en paralelo durante algún tiempo.

- **Fase de mantenimiento:** Genera las correcciones, cambios y mejoras al sistema operativo de información. La fase de mantenimiento inicia cuando un sistema de información se vuelve operacional. La fase de mantenimiento se distingue de las otras fases porque incluye actividades de todas ellas. La fase de mantenimiento termina cuando el desarrollo de un sistema nuevo se justifica mediante el costo. Debido a los altos costos fijos del desarrollo de nuevos sistemas, la fase de mantenimiento puede durar décadas.

El ciclo de vida tradicional ha sido criticado por varias razones. Primero, un sistema operativo no se produce hasta que termina el proceso. Al tiempo de que el sistema es operacional, los requerimientos pudieron haber cambiado. Segundo, generalmente existe cierta prisa por comenzar la implementación para que el producto sea visible. Dada esta prisa, es posible que no se dedique el tiempo suficiente al análisis y el diseño.

Se han propuesto varias metodologías alternativas para solucionar estas dificultades. En las metodologías del desarrollo en espiral, las fases del ciclo de vida se desarrollan para cada uno de los subconjuntos del sistema, que de manera progresiva generan un sistema más grande hasta que el producto final emerge por completo. Las metodologías para el rápido desarrollo de aplicaciones retrasan el diseño de documentos de producción hasta que los requerimientos son claros. Las versiones a menor escala de un sistema, conocidas como **prototipos**, se utilizan para tener en claro los requerimientos. Los prototipos se pueden implementar rápidamente haciendo uso de herramientas gráficas de desarrollo para generar formularios, reportes y código. La implementación de un prototipo les permite a los usuarios proporcionar una significativa retroalimentación a los desarrolladores. Generalmente, los usuarios no comprenden los requerimientos hasta que experimentan con un prototipo. Por lo tanto, los prototipos pueden eliminar los riesgos del desarrollo de un sistema de información, ya que permiten una retroalimentación más directa y oportuna acerca del sistema.

En todas las metodologías de desarrollo se deben generar los modelos gráficos de datos, procesos e interacciones con el entorno. El **modelo de datos** describe los tipos de datos y sus relaciones. El **modelo de procesos** describe las relaciones entre los procesos. Un proceso puede proporcionar datos de entrada que utilicen otros procesos y usar los datos de salida de otros procesos. El **modelo de interacción con el entorno** describe las relaciones entre eventos y procesos. Un evento como el paso del tiempo o una acción del entorno pueden desencadenar que un proceso se inicie o se detenga. La fase del análisis de sistemas genera una versión inicial de estos modelos. La fase de diseño de sistemas agrega más detalles para que los modelos se puedan implementar de forma más eficiente.

Aunque los modelos de datos, procesos e interacciones con el entorno son necesarios para desarrollar un sistema de información, este libro pone énfasis únicamente en los modelos de datos. En muchos esfuerzos de desarrollo de sistemas de información, lo más importante es el modelo de datos. En los sistemas de información empresariales, los modelos de procesos e interacción con el entorno comúnmente se generan después del modelo de datos. En lugar de presentar la notación de los modelos de procesos e interacción con el entorno, este libro se enfoca en los prototipos que ilustran las conexiones entre los datos, los procesos y el entorno. Para conocer más detalles acerca de los modelos de procesos e interacción con el ambiente, por favor consulte las referencias al final del capítulo.

## 2.2 Objetivos del desarrollo de base de datos

En términos generales, el objetivo del desarrollo de bases de datos es crear una base de datos que proporcione un recurso importante a una organización. Para lograr este objetivo, la base de datos debe dar servicio a una gran comunidad de usuarios, apoyar políticas corporativas, incluir datos de alta calidad y ofrecer un acceso eficiente. El resto de esta sección describe con mayor detalle los objetivos del desarrollo de una base de datos.

### **2.2.1 Desarrollar un vocabulario común**

Una base de datos proporciona un vocabulario común a la organización. Antes de implementar una base de datos, las distintas partes de una organización pueden tener una terminología distinta. Por ejemplo, puede haber múltiples formatos para las direcciones, múltiples formas de identificar a los clientes y distintas maneras de calcular las tasas de interés. Después de implementar una base de datos se puede mejorar la comunicación entre las distintas partes de la organización; por lo tanto, una base de datos puede unir a la organización estableciendo un vocabulario común.

Conseguir este vocabulario común no es sencillo. El desarrollo de una base de datos requiere el compromiso de satisfacer a una gran comunidad de usuarios. En cierto sentido un buen diseñador de bases de datos comparte algunas características con un buen político. Un buen político generalmente encuentra soluciones con las que no todo el mundo puede estar de acuerdo. Al establecer un vocabulario común, el diseñador de base de datos también encuentra este tipo de soluciones imperfectas. Los acuerdos pueden ser difíciles, pero los resultados pueden mejorar la productividad, la satisfacción del cliente y otras medidas de desempeño organizacional.

### **2.2.2 Definición del significado de los datos**

Una base de datos contiene reglas de negocio para soportar las políticas organizacionales. La definición de las reglas de negocio es la esencia de la definición de la semántica o significado de una base de datos. Por ejemplo, en un sistema de captura de órdenes, una regla importante es que la orden debe anteceder al embarque. La base de datos puede incluir una restricción de integridad que apoye esta regla. La definición de las reglas de negocio permite que la base de datos apoye de forma activa las políticas organizacionales. Este rol activo contrasta con el rol pasivo que las bases de datos tienen al establecer un vocabulario común.

Al establecer el significado de los datos, un diseñador de bases de datos debe elegir los niveles de restricción apropiados. La selección de los niveles de restricción apropiados puede requerir balancear las necesidades de los distintos grupos. Las restricciones que sean muy estrictas pueden forzar soluciones alternas para manejar las excepciones. En contraste, las restricciones que sean muy flexibles pueden permitir la captura incorrecta de datos dentro de la base de datos. Por ejemplo, en una base de datos universitaria un diseñador debe escoger si un curso se puede almacenar sin saber quién es el instructor. Algunos grupos de usuarios querrán que el instructor sea capturado al inicio para asegurarse de que los compromisos del curso se cumplan. Otros grupos de usuarios desearán mayor flexibilidad, ya que los catálogos de cursos generalmente se imprimen como avance del principio de un periodo académico. Forzar la captura del instructor al momento en que se almacena el curso puede ser muy estricto. Si la base de datos contiene esta restricción, los usuarios pueden verse forzados a darle la vuelta utilizando un valor por omisión, tal como TBA (se anunciará). La restricción apropiada (forzando la captura del instructor o permitiendo su captura posterior) depende de la importancia de las necesidades de los grupos de usuarios con respecto a los objetivos de la organización.

### **2.2.3 Asegurar la calidad de los datos**

La importancia de la calidad de los datos es similar a la importancia de la calidad del producto en una industria. La pobre calidad de un producto puede conducir a pérdidas en las ventas, demandas e insatisfacción del cliente. Como los datos son el producto de un sistema de información, su calidad debe ser igualmente importante. La poca calidad de los datos puede conducir a una deficiente toma de decisiones sobre la comunicación con los clientes, la identificación de clientes repetidos, el rastreo de ventas y la resolución de problemas con los clientes. Por ejemplo, la comunicación con los clientes puede dificultarse si las direcciones no están actualizadas o los nombres de los clientes están asentados de forma inconsistente en diferentes órdenes.

La calidad de los datos tiene muchas dimensiones o características, tal como se ilustra en la tabla 2.1. La importancia de las características de calidad de los datos depende, en gran parte, de la base de datos en la que se apliquen. Por ejemplo, en la descripción del producto en la base de datos de una tienda de abarrotes, puede ser que algunas características importantes de la calidad de los datos sean la caducidad y consistencia de los precios, mientras que para otras partes de la base de datos puede haber otras características más importantes.

**TABLA 2.1**  
**Características comunes de la calidad de datos**

Característica	Significado
Completos	La base de datos representa todos los datos importantes del sistema de información
Faltos de ambigüedad	Cada parte de la base de datos tiene únicamente un significado
Correctos	La base de datos contiene los valores que el usuario percibe
Actualizados	Los cambios del negocio se actualizan en la base de datos sin retrasos excesivos
Confiables	Las fallas o interferencias no corrompen la base de datos
Consistentes	Las distintas partes de la base de datos no están en conflicto

El diseño de una base de datos debe alcanzar una adecuada calidad de datos. Cuando se evalúan las alternativas, el diseñador de base de datos debe considerar las características de calidad para los datos. Por ejemplo, en una base de datos de clientes, el diseñador de base de datos debe considerar la posibilidad de que algunos clientes no tengan una dirección en Estados Unidos. Por lo tanto, el diseño de la base de datos estará incompleto si no contempla las direcciones que no sean de Estados Unidos.

Para conseguir la adecuada calidad de los datos se puede requerir de un balance costo-beneficio. Por ejemplo, en una base de datos de una tienda de abarrotes, los beneficios de la actualización de los precios en el tiempo adecuado se traducen en menos quejas de los consumidores y menos pérdidas por multas del gobierno. Obtener calidad en los datos puede ser costoso, tanto en las actividades preventivas como en las de monitoreo. Por ejemplo, para mejorar la disponibilidad y la actualización de precios, se puede hacer uso de la captura automatizada de datos (actividad preventiva) y del muestreo de la exactitud de los precios al consumidor (actividad de monitoreo).

La interacción costo-beneficio en la calidad de datos debe considerar los costos y los beneficios a largo y corto plazos. Generalmente, los beneficios de la calidad de datos son a largo plazo, en especial aquellos que se intercambian entre bases de datos individuales. Por ejemplo, la coherencia en la identificación de un cliente entre varias bases de datos puede ser un punto crucial para la toma estratégica de decisiones. Dicho punto puede no ser importante para las bases de datos en lo individual. El capítulo 16 trata sobre data warehouses y contempla los elementos de la calidad de datos en relación con la toma estratégica de decisiones.

#### 2.2.4 En busca de la implementación eficiente

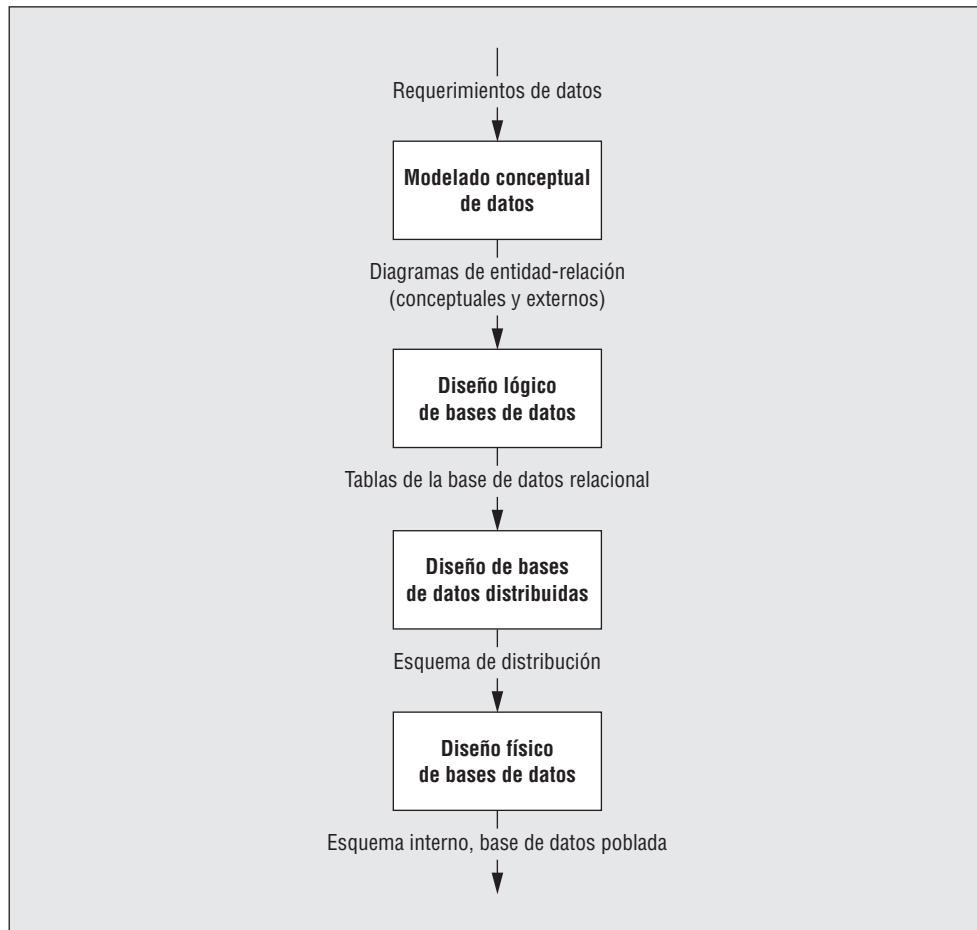
Aun cuando se cumplan las metas del diseño, no se debe utilizar una base de datos con un desempeño pobre. Es importante encontrar una implementación eficiente; sin embargo, una implementación eficiente debe respetar las otras metas tanto como sea posible. Los usuarios de la base de datos deben rechazar la implementación eficiente que comprometa el significado o la calidad de la base de datos.

Encontrar una implementación eficiente es un problema de optimización con un objetivo y restricciones. De manera informal, el objetivo es maximizar el tema del desempeño contra las restricciones del uso de los recursos, y la calidad y significado de los datos. Encontrar una implementación eficiente puede ser difícil por el número de opciones disponibles, la interacción entre las alternativas y la dificultad para describir los accesos. Adicionalmente, encontrar una implementación eficiente requiere de un esfuerzo continuo. El desempeño debe monitorearse y los cambios al diseño deben llevarse a cabo sólo si están garantizados.

### 2.3 Proceso de desarrollo de la base de datos

Esta sección describe las fases del proceso de desarrollo de base de datos y describe su relación con el proceso de desarrollo de los sistemas de información. Los capítulos de las partes 3 y 4 se elaboraron a partir del marco de trabajo que aquí se establece.

**FIGURA 2.3**  
Fases del desarrollo de bases de datos



### 2.3.1 Fases del desarrollo de base de datos

La meta del proceso de desarrollo de base de datos es generar una base de datos operacional para un sistema de información. Para generar una base de datos operacional, usted necesita definir tres esquemas (externo, conceptual e interno) y poblar (proporcionar los datos) la base de datos. Para crear estos esquemas puede seguir el proceso ilustrado en la figura 2.3. Las primeras dos fases se enfocan en la información contenida en la base de datos, mientras que las últimas dos se enfocan en una implementación eficiente. Estas fases se describen con mayor detalle en el resto de esta sección.

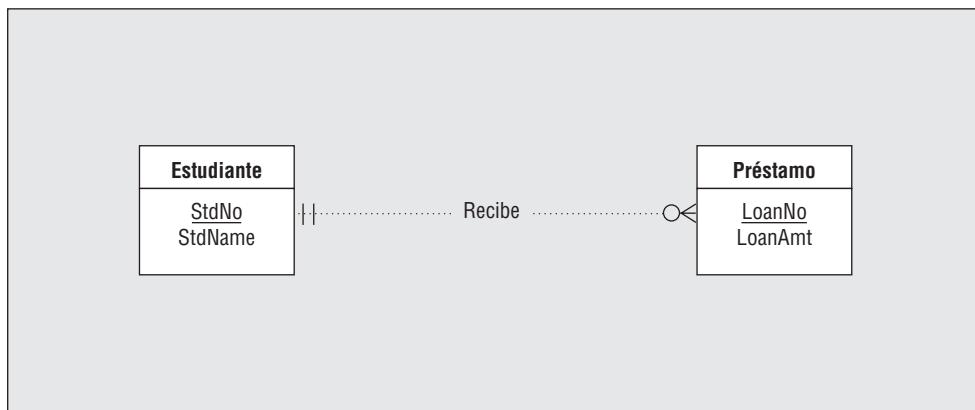
#### *Modelado conceptual de los datos*

La fase del modelado conceptual de los datos utiliza los requerimientos de datos y genera los diagramas entidad-relación (ERD) para el esquema conceptual y para cada uno de los esquemas externos. Los requerimientos de datos pueden tener varios formatos, tales como entrevistas con los usuarios, documentación de los sistemas actuales y formularios y reportes propuestos. El esquema conceptual debe representar todos los requerimientos y formatos. En contraste, los esquemas externos (o vistas) representan los requerimientos de un uso particular de la base de datos, tal como un formulario o reporte en lugar de todos los requerimientos. Por lo tanto, los esquemas externos generalmente son mucho más pequeños que el esquema conceptual.

Los esquemas conceptuales y externos siguen las reglas del modelo entidad-relación, una representación gráfica que ilustra las cosas de interés (entidades) y las relaciones entre ellas. La figura 2.4 muestra un diagrama entidad-relación (ERD) como parte de un sistema de préstamos a estudiantes. Los rectángulos (*Student* y *Loan*) representan los tipos de entidad, y las líneas con leyendas (*Receives*) representan relaciones. Los atributos o propiedades de las entidades se

**FIGURA 2.4**

**ERD parcial del sistema de préstamos a estudiantes**

**FIGURA 2.5**

**Conversión de la figura 2.4**

```

CREATE TABLE Student
(
    StdNo           INTEGER      NOT NULL,
    StdName         CHAR(50),
    ...
    PRIMARY KEY     (StdNo)
)
CREATE TABLE Loan
(
    LoanNo          INTEGER      NOT NULL,
    LoanAmt         DECIMAL(10, 2),
    StdNo           INTEGER      NOT NULL,
    ...
    PRIMARY KEY     (LoanNo),
    FOREIGN KEY     (StdNo) REFERENCES Student
)
    
```

enlistan dentro del rectángulo. El atributo subrayado, conocido como llave primaria, proporciona una identificación única para el tipo de entidad. El capítulo 3 proporciona una definición precisa de las llaves primarias. Los capítulos 5 y 6 presentan más detalles acerca del modelo entidad-relación. Ya que el modelo entidad-relación no se incluye por completo en ningún DBMS, el esquema conceptual no se enfoca en ningún DBMS en específico.

#### *Diseño lógico de bases de datos*

La fase del diseño lógico de las bases de datos transforma el modelo conceptual de datos en un formato comprensible para un DBMS comercial. La fase del diseño lógico no se enfoca en una implementación eficiente. En su lugar, la fase del diseño lógico se enfoca en refinar el modelo conceptual de datos. Los refinamientos preservan el contenido de la información del modelo conceptual de datos mientras que habilitan la implementación en un DBMS comercial. Debido a que la mayoría de las bases de datos corporativas están implementadas en DBMS relacionales, la fase del diseño lógico generalmente produce un diseño de tablas.

La fase del diseño lógico de la base de datos está formada por dos actividades de refinamiento de datos: la conversión y la normalización. La actividad de conversión transforma los ERD en diseños de tablas haciendo uso de reglas de conversión. Tal como aprenderá en el capítulo 3, un diseño de tablas incluye tablas, columnas, llaves primarias, llaves foráneas (enlaces a otras tablas relacionadas) y otras propiedades. Por ejemplo, el ERD de la figura 2.4 se convierte en dos tablas tal como se ilustra en la figura 2.5. La actividad de normalización elimina las redundancias en un diseño de tablas utilizando las restricciones o dependencias entre las columnas. El capítulo 6 presenta las reglas de conversión mientras que el capítulo 7 presenta las técnicas de normalización.

### *Diseño de bases de datos distribuidas*

La fase del diseño de bases de datos distribuidas marca el arranque de las dos primeras fases. El diseño de bases de datos distribuidas y el diseño físico de bases de datos se encuentran enfocados en la implementación eficiente. En contraste, las dos primeras fases (el modelo conceptual de los datos y el diseño lógico de bases de datos) están enfocadas en la información contenida en ellas.

El diseño de bases de datos distribuidas involucra seleccionar la ubicación de los datos y procesos, de tal forma que mejore el desempeño. El desempeño se puede medir de muchas formas, tales como la reducción en los tiempos de respuesta, la disponibilidad de los datos y un mejor control. Para las decisiones sobre la ubicación de los datos, la base de datos se puede dividir de muchas formas para distribuirla entre varios centros de cómputo. Por ejemplo, una tabla de préstamos se puede distribuir de acuerdo con la ubicación del banco que otorga el préstamo. Otra técnica para mejorar el desempeño es replicar o hacer copia de las partes de una base de datos. La replicación mejora la disponibilidad de la base de datos pero hace que las actualizaciones sean más difíciles, ya que varias copias deberán seguir siendo consistentes.

Para las decisiones sobre la ubicación de procesos, una parte del trabajo habitualmente se lleva a cabo en un servidor y otra parte del trabajo se desarrolla en algún cliente. Por ejemplo, el servidor por lo general obtiene los datos y los envía al cliente. El cliente despliega los resultados de una forma atractiva. Existen muchas otras opciones sobre la ubicación de los datos y su procesamiento que se exploran en el capítulo 17.

### *Diseño físico de bases de datos*

La fase del diseño físico de bases de datos, semejante a la fase del diseño de bases de datos distribuidas, se enfoca en una implementación eficiente. Contrario al diseño de bases de datos distribuidas, el diseño físico de bases de datos se enfoca en el desempeño de una sola ubicación de cómputo. Si una base de datos es distribuida, debe decidirse por el diseño físico para cada ubicación. Una implementación eficiente minimiza el tiempo de respuesta sin tener que usar demasiados recursos, como el espacio en disco y memoria. Como los tiempos de respuesta son difíciles de medir de forma directa, se deben sustituir con otras formas de medida como las actividades de entrada/salida hacia el disco.

En la fase de diseño físico de bases de datos existen dos importantes alternativas acerca de la ubicación de los índices y datos. Un índice es un archivo auxiliar que puede ayudar a mejorar el desempeño. Para cada una de las columnas de la tabla el diseñador decide si un índice puede mejorar su desempeño. Un índice puede mejorar el desempeño en la recuperación de datos, pero afectar el desempeño en las actualizaciones. Por ejemplo, pueden mejorar el desempeño los índices sobre una llave primaria (*StdNo* y *LoanNo* de la figura 2.5). En lo que respecta a la ubicación de los datos, el diseñador decide cómo deben agruparse o localizarse en el disco. Por ejemplo, el desempeño puede mejorar si los renglones de los estudiantes se colocan cerca de los renglones asociados con los préstamos. El capítulo 8 describe los detalles del diseño físico de bases de datos, incluyendo la selección de índices y la ubicación de los datos.

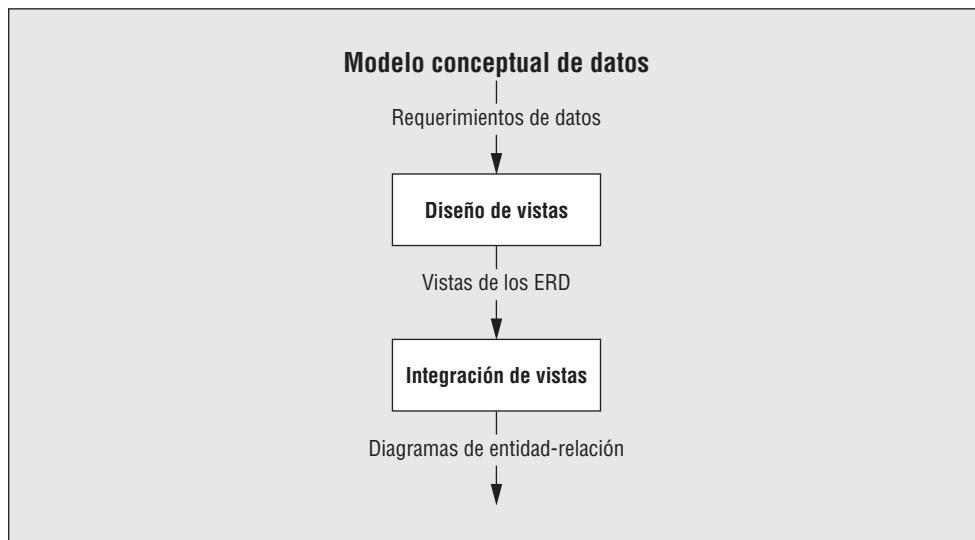
### *Diseño conceptual robusto en grandes proyectos*

El proceso de desarrollo de bases de datos mostrado en la figura 2.3 funciona bien para bases de datos de tamaño moderado. Para bases de datos más grandes, generalmente se modifica la fase del modelado conceptual. El diseño de grandes bases de datos es un proceso que consume tiempo y requiere el trabajo intensivo de todo un equipo de diseñadores. El esfuerzo para este tipo de desarrollo puede involucrar requerimientos provenientes de distintos grupos de usuarios. Para administrar su complejidad, en muchas áreas de la computación se utiliza la estrategia de “divide y vencerás”. La división de un problema grande en partes más pequeñas permite que los problemas pequeños se resuelvan de forma independiente. La solución a los pequeños problemas se combina para tener una solución completa del problema.

El diseño de vistas y la integración (figura 2.6) es una aproximación a la administración de bases de datos grandes y complejas. En el diseño de vistas se construye un ERD por cada grupo de usuarios. Generalmente, una vista es lo suficientemente pequeña como para que una sola persona pueda diseñarla. Varios diseñadores pueden trabajar en vistas que cubran varias partes

**FIGURA 2.6**

**Fragmentación del modelo de datos conceptual en el diseño e integración de vistas**



de la base de datos. El proceso de integración de vistas las combina todas en un único esquema conceptual. La integración involucra el reconocimiento y la resolución de conflictos. Para resolver los conflictos, algunas veces es necesario revisar las vistas en conflicto. El compromiso es una de las partes importantes en la solución de conflictos del proceso de integración de vistas. El capítulo 12 ofrece los detalles acerca del diseño de vistas y del proceso de integración de vistas.

#### *Revisión cruzada con desarrollo de aplicaciones*

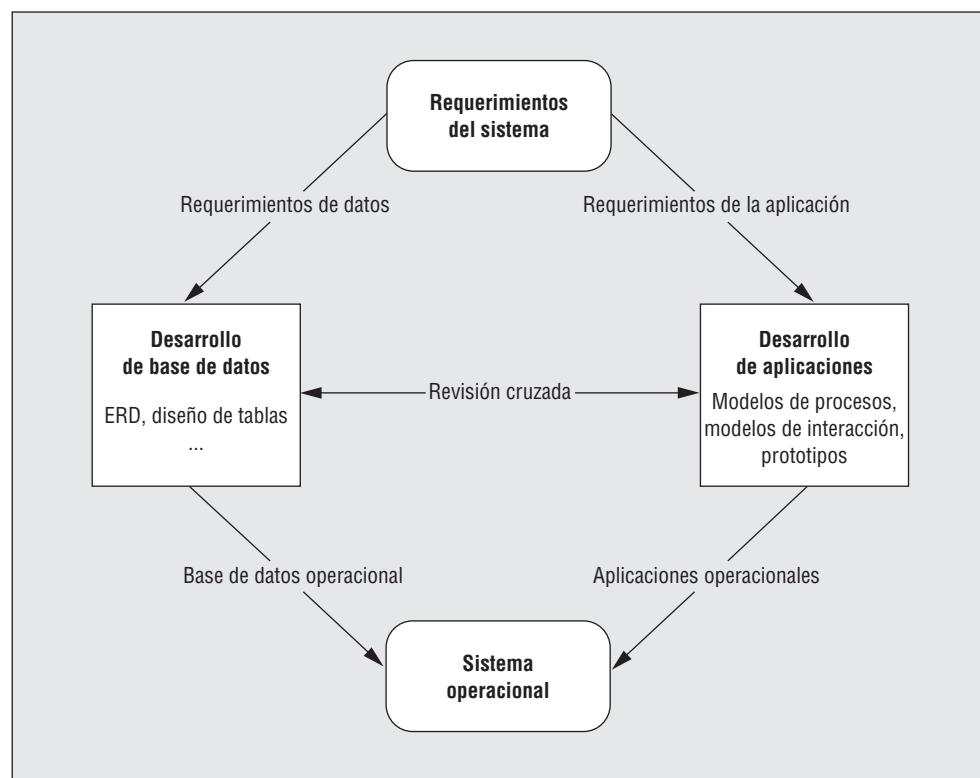
El proceso de desarrollo de bases de datos no existe de forma aislada, sino que se lleva a cabo de forma conjunta con otras actividades del análisis, diseño e implementación de sistemas. La fase del modelado conceptual de datos se realiza como parte de la fase de análisis de sistemas. La fase del diseño lógico de bases de datos se realiza durante el diseño de sistemas. Las fases del diseño de bases de datos distribuidas y del diseño físico de bases de datos comúnmente se encuentran divididas en las fases de diseño e implementación de sistemas. Muchas de las decisiones preliminares para las últimas dos fases se pueden hacer durante el diseño de sistemas; sin embargo, muchas de las decisiones del diseño físico y la distribución deben probarse en una base de datos poblada. Por lo tanto, algunas de las actividades de las últimas dos fases ocurren durante la implementación del sistema.

Para completar los objetivos del desarrollo de bases de datos, este proceso debe acoplarse de forma directa con las otras partes del desarrollo de sistemas de información. Para generar los modelos de datos, procesos e interacción con el ambiente de manera consistente y completa, se puede llevar a cabo una revisión cruzada, como se ilustra en la figura 2.7. El proceso de desarrollo de los sistemas de información se puede dividir entre el desarrollo de las bases de datos y el desarrollo de las aplicaciones. El proceso de desarrollo de bases de datos genera ERD, diseño de tablas, etc., como se describió en esta sección. El proceso de desarrollo de las aplicaciones genera modelos de procesos, interacción y prototipos. Los prototipos son muy importantes al momento de realizar una revisión cruzada. Una base de datos no tiene ningún valor a menos que soporte las aplicaciones para las cuales se construyó, como formularios y reportes. Los prototipos pueden ayudar a revelar las incongruencias entre la base de datos y las aplicaciones que utiliza.

### **2.3.2 Habilidades en el desarrollo de bases de datos**

Como diseñador de bases de datos, usted necesita dos tipos de habilidades, como se ilustra en la figura 2.8. Las fases del modelado conceptuales de datos y del diseño lógico de bases de datos involucran en su mayoría habilidades básicas. Las habilidades básicas son cualitativas, subjetivas y orientadas hacia las personas. Las habilidades cualitativas enfatizan la generación de alternativas viables en lugar de aquellas que son mejores. Como diseñador de bases de datos,

**FIGURA 2.7**  
Interacción entre la base de datos y el desarrollo de aplicaciones

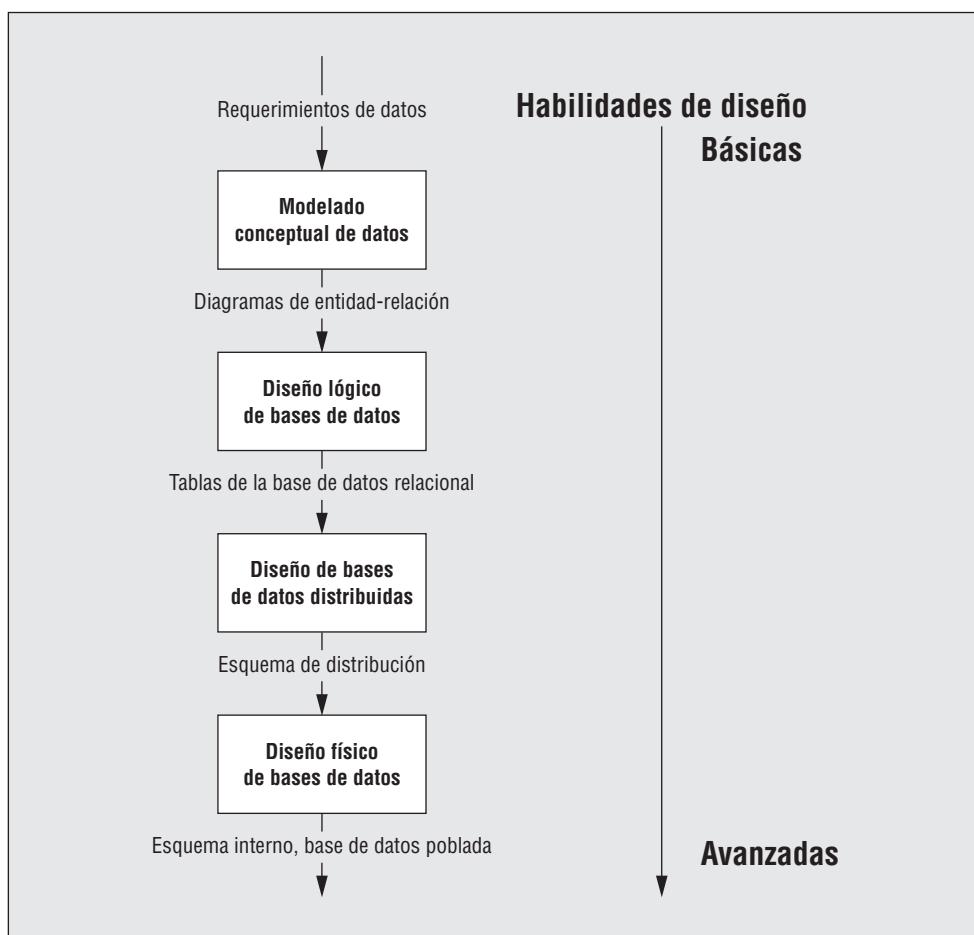


usted querrá generar un rango de alternativas factibles. La elección entre alternativas factibles puede ser subjetiva. Debe identificar los supuestos en los que cada una de las alternativas factibles es la preferida. Generalmente la alternativa seleccionada tiene una base subjetiva a partir del enunciado de otro diseñador sobre los supuestos más viables. El modelado conceptual de datos se orienta hacia las personas. En el papel de modelador de datos, necesita obtener los requerimientos de diversos grupos de usuarios. Como se mencionó previamente, el compromiso y la atención efectiva constituyen habilidades esenciales en el modelado de datos.

El diseño de bases de datos distribuidas y el diseño físico de bases de datos en su mayor parte involucran habilidades avanzadas. Las habilidades avanzadas son cuantitativas, objetivas e intensivas sobre los datos. Puede resultar de gran utilidad algún tipo de conocimiento previo en disciplinas cuantitativas, como la estadística y la administración de operaciones, para comprender los modelos matemáticos utilizados en estas fases. Muchas de las decisiones de estas fases se pueden modelar de forma matemática usando alguna función objetiva y sus limitaciones. Por ejemplo, el objetivo funcional para la elección de un índice es minimizar las lecturas y escrituras al disco con restricciones de delimitación sobre la cantidad de espacio del disco y el tiempo de respuesta. Muchas decisiones no se pueden basar únicamente en criterios objetivos dada la incertidumbre acerca del uso de una base de datos. Para resolver la incertidumbre, puede ser útil un análisis intensivo de los datos. El diseñador de base de datos debe reunir y analizar la información para comprender los patrones sobre el uso y el desempeño de la base de datos.

Dadas las diversas habilidades y el conocimiento previo requerido en varias fases del desarrollo de bases de datos, puede darse la especialización en algún rol. En las grandes organizaciones, los roles del diseño de bases de datos se dividen entre modeladores de datos y expertos en el desempeño de bases de datos. Los modeladores de datos se encuentran en su mayoría involucrados con las fases del modelado conceptual de los datos y del diseño lógico de las bases de datos. Los expertos en el desempeño de bases de datos se involucran principalmente en las fases del diseño de bases de datos distribuidas y del diseño físico de bases de datos. Debido a que estos dos roles requieren de distintas habilidades, la misma persona no desempeñará ambas funciones dentro de las grandes organizaciones. En las organizaciones pequeñas la misma persona puede realizar las dos funciones.

**FIGURA 2.8**  
**Habilidades de diseño utilizadas en el desarrollo de bases de datos**



## 2.4 Herramientas para el desarrollo de bases de datos

Para mejorar la productividad en el desarrollo de sistemas de información se han creado herramientas auxiliares para la ingeniería de software (herramientas CASE, *computer-aided software engineering*, por sus siglas en inglés). Las herramientas CASE pueden ayudar a mejorar la productividad de los profesionales de sistemas de información que trabajan en grandes proyectos, así como la de los usuarios finales que trabajan en proyectos pequeños. Diversos estudios proporcionan evidencias sobre las facilidades que otorgan las herramientas CASE en las fases iniciales del desarrollo de sistemas, lo que conduce a bajar los costos, mejorar la calidad y realizar implementaciones más rápidas.

La mayoría de las herramientas CASE apoyan el proceso de desarrollo de las bases de datos. Algunas herramientas CASE incluyen el desarrollo de bases de datos como parte del desarrollo de sistemas de información. Otras herramientas CASE tienen como objetivo diversas fases del desarrollo de bases de datos sin tomar en cuenta otros aspectos del desarrollo de sistemas de información.

Las herramientas CASE generalmente se clasifican como herramientas *front-end* y *back-end*. Las herramientas CASE *front-end* pueden ayudar a los diseñadores a diagramar, analizar y documentar los modelos usados en el proceso de desarrollo de las bases de datos. Las herramientas CASE *back-end* generan prototipos y código que pueden utilizarse para hacer una revisión cruzada entre la base de datos y otros componentes del sistema de información. Esta sección describe con mayor detalle las funciones de las herramientas CASE y demuestra una herramienta CASE comercial: Microsoft Office Visio Professional 2003.

### 2.4.1 Diagramación

La diagramación es la función más importante y la más utilizada de las herramientas CASE. La mayoría de las herramientas CASE proporcionan las figuras predefinidas y las conexiones entre ellas. Las herramientas de conexión generalmente permiten que las figuras se muevan y permanezcan conectadas como si estuvieran “pegadas”. Esta característica del pegado proporciona una flexibilidad importante, ya que los símbolos de los diagramas generalmente se reacomodan muchas veces.

Las herramientas CASE proporcionan varias características para bosquejos más grandes. La mayoría de las herramientas CASE permiten que los diagramas se expandan a lo largo de varias páginas. Los bosquejos que abarcan varias páginas pueden imprimirse de tal forma que las páginas se peguen para crear un mural. El despliegue puede resultar difícil cuando se trata de bosquejos grandes. Algunas herramientas CASE intentan mejorar la apariencia visual de un diagrama mediante la creación de un despliegue automático. El despliegue automático puede minimizar el número de conexiones que se cruzan en un diagrama. Aunque el despliegue automatizado generalmente no es suficiente por sí solo, un diseñador puede utilizarlo como un primer paso para mejorar la apariencia visual de un diagrama grande.

### 2.4.2 Documentación

La documentación es una de las funciones más antiguas y valiosas de las herramientas CASE. Las herramientas CASE pueden almacenar múltiples propiedades de un modelo de datos y ligar las propiedades con símbolos del diagrama. Algunos ejemplos de las propiedades almacenadas en una herramienta CASE incluyen los nombres de alias, reglas de integridad, tipos de datos y dueños. Además de las propiedades, las herramientas CASE pueden almacenar el texto que describa los supuestos, las alternativas y las notas. Al diccionario de datos también se le conoce como repositorio o enciclopedia.

Para apoyar la evolución de los sistemas, muchas herramientas CASE pueden documentar versiones. Una versión es un grupo de cambios o mejoras hechas al sistema que se publican de forma conjunta. Debido al volumen de cambios, por lo general se publican grupos de cambios en vez de los cambios individuales. Durante la vida de un sistema de información se pueden hacer muchas versiones. Muchas herramientas CASE soportan la documentación de cambios individuales y de versiones completas para ayudar a comprender las relaciones entre versiones.

### 2.4.3 Análisis

Las herramientas CASE pueden proporcionar asistencia activa a los diseñadores de bases de datos a través de funciones de análisis. Estas herramientas ayudan a que los diseñadores sean más eficientes en la documentación y la diagramación. En las funciones de análisis, las herramientas CASE pueden realizar el trabajo de un diseñador de bases de datos. Una función de análisis es cualquier forma de razonamiento aplicada a las especificaciones generadas durante el proceso de desarrollo de bases de datos. Por ejemplo, una función de análisis importante es hacer la conversión entre un ERD y un diseño de tablas. A la conversión de un ERD en un diseño de tablas se le conoce como ingeniería progresiva y a la conversión en la dirección contraria se le conoce como ingeniería en reversa.

Las funciones de análisis aplican para cada una de las fases del desarrollo de bases de datos. En la fase del modelado conceptual de datos, las funciones de análisis pueden mostrar conflictos de algún ERD. La normalización elimina la redundancia en el diseño de tablas. En las fases de diseño de bases de datos distribuidas y diseño físico de bases de datos, las funciones de análisis pueden sugerir decisiones acerca de la ubicación de los datos y la selección de índices. Además, las funciones de análisis que controlan las versiones pueden utilizarse entre las fases de desarrollo de bases de datos. Las funciones de análisis pueden hacer conversiones entre las versiones y mostrar una lista de las diferencias entre ellas.

Las funciones de análisis son características avanzadas de las herramientas CASE, por lo que la disponibilidad de las funciones de análisis es muy variada. Algunas herramientas CASE soportan poco o no soportan las funciones de análisis, mientras que otras incluyen un soporte bastante amplio. Debido a que las funciones de análisis pueden ser útiles en cada una de las fases

del desarrollo de bases de datos, ninguna herramienta CASE individual proporciona un rango completo de funciones de análisis. Las herramientas CASE tienden a especializarse de acuerdo con las fases que soportan. Las herramientas CASE que son independientes de algún DBMS típicamente se especializan en las funciones de análisis de la fase del modelado conceptual de datos. En contraste, las herramientas CASE ofrecidas por los fabricantes de DBMS generalmente se especializan en las fases de diseño de bases de datos distribuidas y en el diseño físico de bases de datos.

#### 2.4.4 Herramientas de prototipos

Las herramientas de prototipos proporcionan un enlace entre el desarrollo de bases de datos y el desarrollo de aplicaciones. Las herramientas para crear prototipos se pueden utilizar para crear formularios y reportes haciendo uso de una base de datos. Debido a que las herramientas para crear prototipos pueden generar código (sentencias SQL y código de un lenguaje de programación), algunas veces se les conoce como herramientas generadoras de código. Las herramientas de prototipos por lo general se incluyen como parte del DBMS. Estas herramientas pueden ofrecer asistentes que le ayuden al desarrollador a crear aplicaciones de forma rápida y puedan ser probadas por los usuarios. Las herramientas de prototipos también pueden crear un diseño de bases de datos inicial extrayendo los diseños actuales de alguna biblioteca de diseños. Este tipo de herramienta le puede resultar muy útil a los usuarios finales y a los nuevos diseñadores de bases de datos.

**TABLA 2.2**  
**Herramientas destaca-das CASE para el desarrollo de bases de datos**

Herramienta	Fabricante	Características de innovación
PowerDesigner 10	Sybase	Ingeniería progresiva y en reversa para las bases de datos relacionales y lenguajes de programación varios; soporte para la administración de modelos al compararlos y unirlos; generación de código de aplicaciones; soporte UML; modelado de procesos de negocio; generación de código XML; control de versiones; soporte para el modelado de data warehouse
Oracle Designer 10g	Oracle	Ingeniería progresiva y en reversa para las bases de datos relacionales; ingeniería en reversa de formularios; generación de código de aplicaciones; control de versiones; análisis de dependencias; modelado de los procesos de negocio; análisis de referencias cruzadas
Visual Studio .Net Enterprise Architect	Microsoft	Ingeniería progresiva y en reversa para las bases de datos relacionales y para el lenguaje de modelado unificado (UML); generación de código para servicios web XML; soporte para las guías de arquitectura; generación de modelos de datos a partir de las descripciones del lenguaje natural
AllFusion ERWin Data Modeler	Computer Associates	Ingeniería progresiva y en reversa para las bases de datos relacionales; generación de código de aplicaciones; soporte al modelado de data warehouse; herramientas para la reutilización de modelos
ER/Studio 6.6	Embarcadero Technologies	Ingeniería progresiva y en reversa para las bases de datos relacionales; generación de código en Java u otro lenguaje; soporte para la administración de modelos para compararlos y unirlos; soporte para UML; control de versiones; soporte para la administración de DBMS múltiples
Visible Analyst 7.6	Visible Systems Corporation	Ingeniería progresiva y en reversa para las bases de datos relacionales; soporte para la administración de modelos para compararlos y unirlos; control de versiones; soporte para la revisión de reglas y metodología; soporte para la planeación estratégica

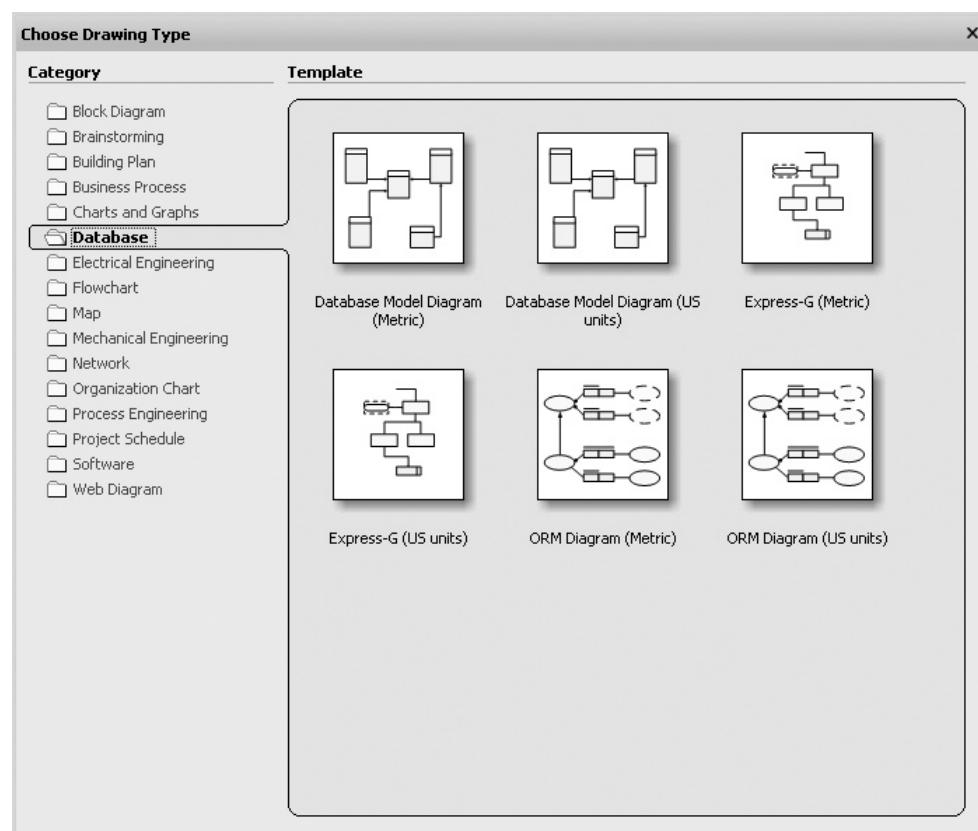
### 2.4.5 Herramientas CASE comerciales

Tal como se muestra en la tabla 2.2, existen varias herramientas CASE que proporcionan múltiples funcionalidades para el desarrollo de bases de datos. Cada uno de los productos de la tabla 2.2 soporta el ciclo de vida completo del desarrollo de sistemas de información, aunque la calidad, profundidad y rango de las características es variable entre ellos. Además, la mayoría de los productos de la tabla 2.2 tienen distintas versiones que varían en precio y características. Todos los productos son relativamente neutrales con respecto a un DBMS en particular, aunque cuatro de los productos son ofrecidos por organizaciones que poseen la mayoría de los productos DBMS. Además de las características de los productos de la tabla 2.2, otras compañías ofrecen herramientas CASE que se especializan en un subconjunto de las fases de desarrollo de bases de datos.

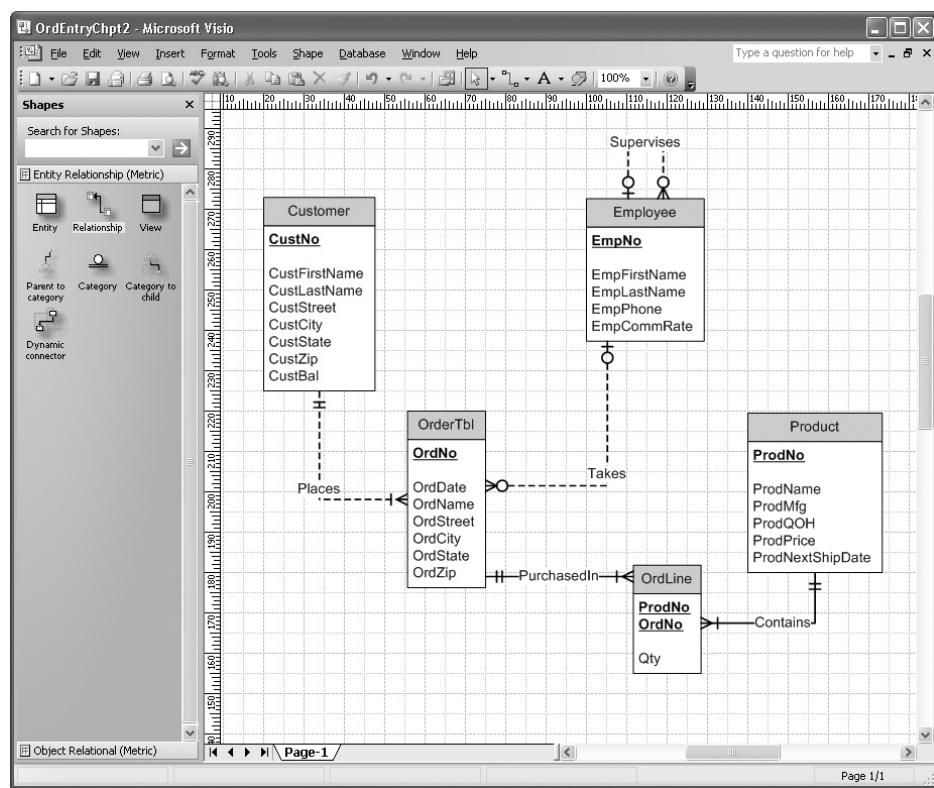
Para proporcionar una idea de algunas de las características de las herramientas CASE comerciales, se ilustra de forma breve Microsoft Office Visio 2003 Professional, una versión básica de Visual Studio .Net Enterprise Architect. Visio Professional proporciona excelentes capacidades para dibujar y un número importante de herramientas útiles para el análisis. Esta sección ilustra Visio Professional porque es una herramienta poderosa y fácil de usar en los cursos de introducción de bases de datos.

Para el desarrollo de bases de datos, Visio Professional incluye varias plantillas (conjuntos de figuras) y el soporte de diccionarios de datos. Tal como se muestra en la figura 2.9, Visio proporciona plantillas para distintas notaciones del modelado de datos [Database Model Diagram, Express-G y Object Role Modeling (ORM)], así como para el Lenguaje de Modelado Unificado (disponible en la carpeta de software). La figura 2.10 ilustra la plantilla entidad-relación (a la izquierda) y la ventana para dibujar (a la derecha). Si se mueve alguno de los símbolos, permanece conectado con los otros símbolos gracias a la característica de “pegado”. Por ejemplo, si el rectángulo de producto se mueve, permanece conectado al rectángulo OrdLine a través de la línea PurchaseIn. Visio Professional puede modificar de forma automática el diagrama por completo si así se le solicita.

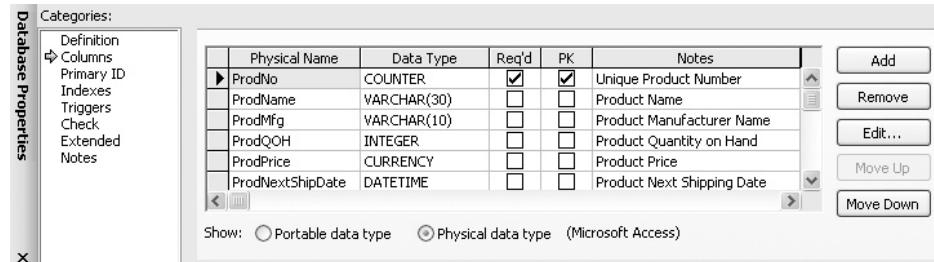
**FIGURA 2.9**  
Plantillas para el  
modelado de datos  
en Visio 2003 Profes-  
sional



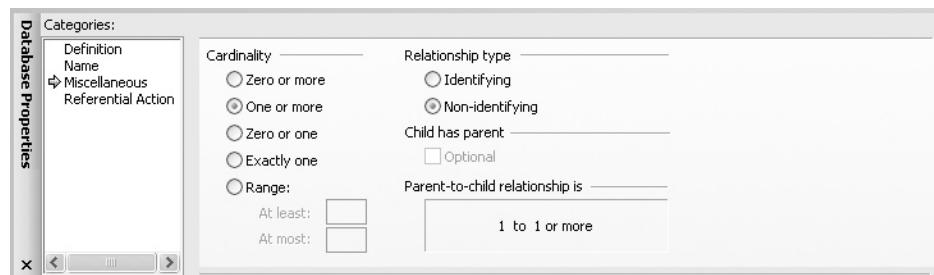
**FIGURA 2.10**  
Plantillas y ventanas de dibujo en Visio Professional



**FIGURA 2.11**  
Ventana de propiedades de base de datos en Visio Professional para el tipo de entidad *Product*



**FIGURA 2.12**  
Ventana de propiedades de base de datos en Visio Professional para la relación *Places*



Visio proporciona un diccionario de datos para acompañar la plantilla de entidad-relación. Para los tipos de entidad (símbolos de rectángulo), Visio provee el nombre, tipo de dato, llave primaria requerida (PK) e identificación de propiedades, tal como se muestra en las columnas de categoría de la figura 2.11, así como otras propiedades de las categorías que no han sido seleccionadas. Para las relaciones (líneas que conectan los símbolos), Visio soporta propiedades sobre la definición, nombre, cardinalidad y acciones referenciales, como se ilustra en la figura 2.12. Para un soporte al diccionario adicional de datos pueden agregarse propiedades personalizadas o específicas de algún DBMS.

Visio proporciona diversas herramientas de análisis y de creación de prototipos, adicionales a las características de las plantillas y del diccionario. Las herramientas de análisis dan soporte principalmente a la tarea de conversión de esquemas de la fase del diseño lógico de la base de datos. El asistente para refrescar el modelo detecta y soluciona las diferencias entre un diagrama de bases de datos de Visio y una base de datos relacional existente. El asistente de ingeniería en reversa realiza diversas tareas para convertir la definición de una base de datos relacional en un diagrama de bases de datos de Visio. Visio también soporta varias revisiones de errores para asegurar la consistencia en los diagramas de bases de datos. Para crear prototipos, Visio puede almacenar las figuras en bases de datos relacionales. Esta característica puede ser muy útil para proporcionar una interfase visual para los datos jerárquicos, tales como organigramas y datos de facturación de materiales. Para crear prototipos más potentes, Visio soporta el lenguaje Visual Basic with Applications (VBA), un lenguaje basado en eventos que está integrado con Microsoft Office.

## **Reflexión final**

Al inicio de este capítulo se describió el rol de las bases de datos dentro de los sistemas de información y la naturaleza del proceso de desarrollo de bases de datos. Los sistemas de información son conjuntos de componentes relacionados que generan datos para la toma de decisiones. Las bases de datos representan la memoria permanente de los sistemas de información. El desarrollo de un sistema de información involucra un proceso repetitivo de análisis, diseño e implementación. El desarrollo de bases de datos ocurre en todas las fases del desarrollo de sistemas. Debido a que una base de datos generalmente es parte crucial de un sistema de información, el desarrollo de bases de datos puede ser la parte dominante en el desarrollo de dichos sistemas. El desarrollo de los componentes de procesamiento e interacción generalmente se lleva a cabo después del desarrollo de la base de datos. La revisión cruzada entre la base de datos y las aplicaciones es la liga que conecta el proceso de desarrollo de bases de datos con el proceso de desarrollo de los sistemas de información.

Una vez presentado el rol de las bases de datos y la naturaleza del desarrollo de las bases de datos, este capítulo describe metas, fases y herramientas para el desarrollo de bases de datos. Las metas enfatizan tanto el contenido informativo de la base de datos como su implementación eficiente. Las fases del desarrollo de una base de datos establecen en primer lugar el contenido informativo de la base de datos y después buscan una implementación eficiente. Las fases del modelado conceptual de datos y del diseño lógico de bases de datos involucran el contenido informativo de la base de datos. En la implementación eficiente se involucran las fases del diseño de bases de datos distribuidas y el diseño físico de bases de datos. Debido al reto que implica el proceso de desarrollo de bases de datos, se han creado herramientas de ingeniería de software asistidas por computadora (CASE) para mejorar la productividad. Las herramientas CASE pueden ser esenciales para ayudar al diseñador de bases de datos a trazar, documentar y crear el prototipo de una base de datos. Además, algunas herramientas CASE proporcionan asistencia activa en el análisis del diseño de la base de datos.

Este capítulo le ofrece un contexto para los capítulos de las partes 3 y 4; incluso puede ser que usted quiera volver a éste después de completar dichos capítulos. Los capítulos de las partes 3 y 4 proporcionan detalles acerca de las fases del desarrollo de bases de datos. Los capítulos 5 y 6 presentan detalles del modelo entidad-relación, la práctica del modelado de datos usando el modelo entidad-relación y la conversión del modelo de entidad-relación al modelo relacional. El capítulo 7 presenta las técnicas de normalización para tablas relacionales, y el capítulo 8 presenta las técnicas del diseño físico de bases de datos.

## **Revisión de conceptos**

- Sistema: componentes relacionados que trabajan en conjunto para cumplir objetivos.
- Sistema de información: sistema que acepta, procesa y genera datos.
- Modelo en cascada del desarrollo de sistemas de información: marco de referencia para las actividades del proceso de desarrollo de sistemas de información.
- Metodologías de desarrollo en espiral y de desarrollo rápido de aplicaciones para solucionar los problemas del enfoque tradicional en el desarrollo en cascada.

- Rol de las bases de datos dentro de los sistemas de información: proporcionan memoria permanente.
- Definir un vocabulario en común para unificar a una organización.
- Definir las reglas de negocio que soportan los procesos organizacionales.
- Asegurar la calidad de los datos para mejorar la calidad en la toma de decisiones.
- Evaluar la inversión en la calidad de datos usando un enfoque costo-beneficio.
- Encontrar una implementación eficiente para asegurar el desempeño adecuado sin comprometer otras metas del diseño.
- Modelado de datos conceptual para representar el contenido de la información, independiente del DBMS de destino.
- Diseño e integración de vistas para administrar la complejidad de los esfuerzos del modelo de datos a gran escala.
- Diseño lógico de bases de datos para refinar un modelo de datos conceptual para un DBMS específico.
- Diseño de bases de datos distribuidas para determinar la ubicación de los datos y procesos con el fin de obtener una implementación eficiente y confiable.
- Diseño físico de bases de datos para lograr implementaciones eficientes en cada lugar de cómputo.
- Desarrollo de prototipos de formularios y reportes para hacer una revisión cruzada entre la base de datos y las aplicaciones que la utilizan.
- Habilidades básicas para el modelado de datos conceptual: cualitativas, subjetivas y orientadas hacia las personas.
- Habilidades avanzadas para una implementación eficiente: cuantitativas, objetivas y enfocadas a los datos.
- Herramientas de ingeniería de software asistidas por computadora (CASE) para mejorar la productividad del proceso de desarrollo de bases de datos.
- Ayuda fundamental de las herramientas CASE: dibujar y documentar.
- Ayuda activa de las herramientas CASE: análisis y creación de prototipos.

## Preguntas

1. ¿Cuál es la relación entre un sistema y un sistema de información?
2. Proporcione un ejemplo de un sistema que no sea un sistema de información.
3. Para un sistema de información que conozca, describa algunos de sus componentes (datos de entrada, datos de salida, personas, software, hardware y procedimientos).
4. Describa brevemente algunos de los tipos de datos de una base de datos para el sistema de información de la pregunta 3.
5. Describa las fases del modelo de cascada.
6. ¿Por qué se le considera al modelo de cascada únicamente como un marco de trabajo de referencia?
7. ¿Cuáles son las desventajas del modelo de cascada?
8. ¿Qué metodologías alternas se han propuesto para solucionar las dificultades del modelo de cascada?
9. ¿Cuál es la relación entre el proceso de desarrollo de bases de datos y el proceso de desarrollo de sistemas de información?
10. ¿Qué es un modelo de datos? ¿Y un modelo de procesos? ¿Y un modelo de interacción con el ambiente?
11. ¿Cuál es el objetivo de crear un prototipo en el proceso de desarrollo de los sistemas de información?
12. ¿Cómo es que el diseñador de base de datos se asemeja a un político al establecer un vocabulario en común?
13. ¿Por qué un diseñador de bases de datos debe establecer el significado de los datos?

14. ¿Qué factores debe de considerar el diseñador de base de datos cuando seleccione las restricciones de una base de datos?
15. ¿Por qué es importante la calidad de los datos?
16. Proporcione ejemplos de problemas de calidad de datos de acuerdo con las dos características mencionadas en la sección 2.2.3.
17. ¿Cómo hace un diseñador de bases de datos para decidir el nivel apropiado de la calidad de los datos?
18. ¿Por qué es importante encontrar una implementación eficiente?
19. ¿Cuáles son las entradas y las salidas de la fase del modelado de datos conceptual?
20. ¿Cuáles son las entradas y salidas de la fase del diseño lógico de bases de datos?
21. ¿Cuáles son las entradas y salidas de la fase del diseño de las bases de datos distribuidas?
22. ¿Cuáles son las entradas y salidas de la fase del diseño físico de bases de datos?
23. ¿Qué significa decir que la fase del modelado de base de datos conceptual y la fase del diseño lógico de las bases de datos se enfocan en el contenido informativo de la base de datos?
24. ¿Por qué hay dos fases (modelado conceptual de datos y diseño lógico de las bases de datos) que involucran el contenido informativo de la base de datos?
25. ¿Cuál es la relación del diseño de vistas y la integración de vistas con el modelado conceptual de datos?
26. ¿Qué es una habilidad básica?
27. ¿Qué fases del desarrollo de bases de datos involucran principalmente habilidades básicas?
28. ¿Qué es una habilidad avanzada?
29. ¿Qué fases del desarrollo de bases de datos involucran principalmente habilidades avanzadas?
30. ¿Qué tipo de antecedente es el apropiado para las habilidades avanzadas?
31. ¿Por qué las grandes organizaciones tienen en algunas ocasiones distintas personas realizando las fases de diseño que tienen que ver con contenido de información e implementación eficiente?
32. ¿Por qué las herramientas CASE son útiles en el proceso de desarrollo de bases de datos?
33. ¿Cuál es la diferencia entre las herramientas CASE *front-end* y *back-end*?
34. ¿Qué tipo de soporte puede proporcionar una herramienta CASE para dibujar un diagrama de bases de datos?
35. ¿Qué tipo de soporte puede proporcionar una herramienta CASE para documentar el diseño de bases de datos?
36. ¿Qué tipo de soporte puede proporcionar una herramienta CASE para analizar el diseño de una base de datos?
37. ¿Qué tipo de soporte puede proporcionar una herramienta CASE para crear prototipos?
38. ¿Usted esperaría encontrar algún fabricante de software que proporcione un amplio rango de funciones (dibujar, documentar, analizar y crear prototipos) para el proceso de desarrollo de bases de datos? ¿Por qué?

## Problemas

Dada la naturaleza introductoria de este capítulo, no existen problemas en él. Los problemas aparecen al final de los capítulos de las partes 3 y 4.

## Referencias para ampliar su estudio

Para una descripción más detallada del proceso de desarrollo de bases de datos puede consultar libros especializados sobre el diseño de bases de datos, como Batini, Ceri y Navathe (1992) y Teorey (1999). Para más detalles del proceso de desarrollo de sistemas, puede consultar libros sobre el análisis y diseño de sistemas, como Whitten y Bentley (2004). Para más información acerca de la calidad de datos, consulte libros especializados sobre la calidad de datos, incluyendo Olson (2002) y Redman (2001).



Parte

# Comprendiendo las bases de datos relacionales

---



Los capítulos de la parte 2 proporcionan una introducción detallada al modelo relacional de datos con el fin de crear los fundamentos para el diseño de bases de datos y el desarrollo de aplicaciones con bases de datos relacionales. El capítulo 3 presenta los conceptos de definición de datos y operadores de recuperación para las bases de datos relacionales. El capítulo 4 muestra la recuperación mediante SQL y las sentencias de modificación para problemas de complejidad baja e intermedia y pone énfasis en las herramientas mentales para desarrollar habilidades para la construcción de consultas (*query*).

---

**Capítulo 3.** El modelo relacional de datos.

**Capítulo 4.** Formulación de consultas con SQL.



# Capítulo 3

---

## El modelo relacional de datos

### Objetivos de aprendizaje

Este capítulo proporciona los fundamentos para usar una base de datos relacional. Al finalizar este capítulo el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Reconocer la terminología de la base de datos relacional.
- Comprender el significado de las reglas de integridad para las bases de datos relacionales.
- Comprender el impacto de las filas referenciadas en el mantenimiento de las bases de datos relacionales.
- Comprender el significado de cada operador de álgebra relacional.
- Listar las tablas que se deben combinar para obtener los resultados deseados en las solicitudes de recuperación simple.

### Panorama general

---

Los capítulos de la parte 1 proporcionaron un punto de partida para la exploración de la tecnología de bases de datos y la comprensión del proceso de desarrollo de bases de datos. Mostramos ampliamente las características de las bases de datos y de los DBMS, los objetivos del desarrollo de las bases de datos y las fases del proceso de su desarrollo. Este capítulo se enfoca especialmente en el modelo relacional de datos. Los DBMS relacionales dominan el mercado de los DBMS de negocio. Sin lugar a dudas usted utilizará los DBMS relacionales a lo largo de su carrera como profesional en sistemas de información. Este capítulo proporciona los fundamentos para que usted sea eficiente al diseñar bases de datos y, en capítulos posteriores, pueda desarrollar aplicaciones de bases de datos relacionales.

Para usar de forma efectiva una base de datos relacional usted necesitará dos tipos de conocimientos. Primero, necesitará entender las estructuras y contenidos de las tablas de la base de datos. Entender las conexiones entre las tablas es especialmente importante, ya que muchas recuperaciones de bases de datos involucran varias tablas. Este capítulo presenta la terminología básica, las reglas de integridad y una notación para visualizar las conexiones entre las tablas, todo con el fin de entender las bases de datos relacionales. En segundo lugar, requiere comprender los operadores de álgebra relacional, ya que son los cimientos de la mayoría de los lenguajes comerciales de consulta. La comprensión de los operadores mejorará su conocimiento acerca de lenguajes de consulta como SQL. Para ayudarlo a comprender el significado de cada operador, este capítulo ofrece una representación visual de cada operador y varios resúmenes.

## 3.1 Elementos básicos

---

Los sistemas de bases de datos relacionales fueron desarrollados inicialmente por su familiaridad y simplicidad. Debido a que las tablas se usan para comunicar ideas en muchos terrenos del conocimiento, la terminología de tablas, filas y columnas no es desconocida del todo para la mayoría de los usuarios. Durante los primeros años de las bases de datos relacionales (década de 1970), la simplicidad y la familiaridad de las bases de datos relacionales fueron sumamente atractivas, en especial al compararse con orientaciones procedurales de otros modelos de datos existentes por entonces. A pesar de la familiaridad y de la simplicidad de las bases de datos relacionales, existe también una fuerte base matemática. Las matemáticas de las bases de datos relacionales incluyen la conceptualización de las tablas como conjuntos. La combinación de la familiaridad y la simplicidad con los fundamentos matemáticos es tan poderosa que los DBMS relacionales dominan comercialmente.

Esta sección presenta la terminología básica de las bases de datos relacionales e introduce la sentencia CREATE TABLE del lenguaje de consulta estructurada (SQL). De la sección 3.2 a la 3.4 se proporcionan más detalles de los elementos definidos en esta sección.

### 3.1.1 Tablas

#### tabla

arreglo bidimensional de datos. Una tabla está formada por un encabezado que define el nombre de la tabla y los nombres de columnas y un cuerpo que contiene las filas de datos.

Una base de datos relacional está formada por una recopilación de tablas. Cada tabla tiene una parte de definición o encabezado y una parte de contenido o cuerpo. La parte del encabezado consiste en el nombre de la tabla y de las columnas. Por ejemplo, una tabla estudiante puede contener columnas para número de seguridad social, nombre, dirección, ciudad, estado, código postal, clase (de reciente ingreso, estudiante de segundo año, etc.), carrera y promedio acumulado. El cuerpo muestra las filas de la tabla. Cada fila en una tabla estudiante representa a un estudiante inscrito en la universidad. Una tabla estudiante para una carrera universitaria puede tener más de 30 000 filas, demasiadas para verlas al mismo tiempo.

Para entender una tabla también es útil revisar algunas de sus filas. Un listado de la tabla u hoja de datos muestra el nombre de las columnas en la primera fila y el cuerpo en las filas restantes. La tabla 3.1 muestra un listado de tablas para la tabla *Student*. Se muestran tres filas de ejemplo que representan a los estudiantes de una universidad. La manera de nombrar las columnas en este libro es usando una abreviación del nombre de la tabla (Std) seguida por un nombre descriptivo. Debido a que los nombres de las columnas generalmente se usan sin identificar las tablas asociadas, la abreviación facilita la asociación sencilla de tablas. Las mayúsculas y minúsculas intercaladas remarcán las diferentes partes del nombre de una columna.

Para definir el encabezado de una tabla se puede usar una sentencia CREATE TABLE. CREATE TABLE es una sentencia de lenguaje de consulta estructurada (SQL). Debido a que SQL es un lenguaje estándar en la industria, en la mayoría de los DBMS se puede usar la sentencia CREATE TABLE para crear tablas. La sentencia CREATE TABLE que se muestra crea la tabla *Student*.<sup>1</sup> Para cada una de las columnas se especifican el nombre de columna y el tipo de dato. Los tipos de datos indican la clase de dato (carácter, numérico, si/no, etc.) y las operaciones permitidas para cada columna (operaciones numéricas, operaciones sobre las cadenas de caracteres, etc.). Cada tipo de dato tiene un nombre (por ejemplo, CHAR para carácter) y,

#### tipo de datos

define un conjunto de valores y sus operaciones. Cada columna de una tabla se asocia con un tipo de dato.

**TABLA 3.1 Ejemplo de listado de tablas para la tabla estudiante**

StdSSN	StdFirstName	StdLastName	StdCity	StdState	StdZip	StdMajor	StdClass	StdGPA
123-45-6789	HOMER	WELLS	SEATTLE	WA	98121-1111	IS	FR	3.00
124-56-7890	BOB	NORBERT	BOTHELL	WA	98011-2121	FIN	JR	2.70
234-56-7890	CANDY	KENDALL	TACOMA	WA	99042-3321	ACCT	JR	3.50

<sup>1</sup> Las sentencias CREATE TABLE de este capítulo corresponden a la sintaxis estándar de SQL. Existen pequeñas diferencias para la mayoría de los DBMS comerciales.

**TABLA 3.2**  
Breve descripción de tipos de datos comunes en SQL

Tipo de dato	Descripción
<i>CHAR(L)</i>	Para las capturas de texto de longitud fija, como las abreviaciones de un estado y los números de seguridad social. Cada valor de la columna para CHAR contiene el número máximo de caracteres ( <i>L</i> ), incluso si la longitud actual es más corta. La mayoría de los DBMS tienen un límite superior aproximado de 255 para la longitud ( <i>L</i> ).
<i>VARCHAR(L)</i>	Para texto de longitud variable como los nombres y las direcciones de calles. Los valores de las columnas que usen VARCHAR únicamente tienen el número actual de caracteres, no la longitud máxima como las columnas CHAR. La mayoría de los DBMS tienen un límite superior aproximado de 255 para la longitud.
<i>FLOAT(P)</i>	Para columnas que contienen datos numéricos con una precisión flotante, como los cálculos de tasas de interés y cálculos científicos. El parámetro de precisión <i>P</i> indica el número de dígitos significativos. La mayoría de los DBMS tienen un límite superior aproximado de 38 para <i>P</i> . Algunos DBMS tienen dos tipos de datos, REAL y DOUBLE PRECISION, para la precisión baja y alta de los números de punto flotante, en lugar de la variable de precisión con el tipo de dato FLOAT.
<i>DATE/TIME</i>	Para las columnas que contienen fechas y horas, como una fecha de pedido. Este tipo de datos no es estándar entre los DBSM. Algunos sistemas soportan tres tipos de datos (DATE, TIME y TIMESTAMP), mientras que otros soportan una combinación de tipos de datos (DATE) que almacenan tanto la fecha como la hora.
<i>DECIMAL(W, R)</i>	Para columnas que contienen datos numéricos con una precisión fija, tal como los montos monetarios. El valor <i>W</i> indica el número total de dígitos y el valor <i>R</i> indica el número de dígitos a la derecha del punto decimal. A este tipo de dato también se le conoce en algunos sistemas como NUMERIC.
<i>INTEGER</i>	Para columnas que contienen números enteros (por ejemplo, números sin punto decimal). Algunos DBMS tienen el tipo de dato SMALLINT para números enteros muy pequeños y el tipo de dato LONG para enteros muy grandes.
<i>BOOLEAN</i>	Para columnas que contienen datos con dos valores, como verdadero/falso o sí/no.

generalmente, una especificación de longitud. La tabla 3.2 enumera tipos de datos comunes que se usan en los DBMS relacionales.<sup>2</sup>

```
CREATE TABLE Student
(
    StdSSN           CHAR(11),
    StdFirstName     VARCHAR(50),
    StdLastName      VARCHAR(50),
    StdCity          VARCHAR(50),
    StdState         CHAR(2),
    StdZip           CHAR(10),
    StdMajor          CHAR(6),
    StdClass          CHAR(6),
    StdGPA            DECIMAL(3,2)
)
```

### 3.1.2 Conexiones entre tablas

#### relación

conexión entre las filas de dos tablas. Las relaciones se muestran mediante los valores de una columna en una tabla que coinciden con los valores de una columna en otra tabla.

No es suficiente con entender cada tabla en lo individual. Para comprender una base de datos relacional también se deben comprender las conexiones o relaciones entre las tablas. Las filas de una tabla generalmente se relacionan con las filas de otras tablas. Los valores que coinciden (idénticos) indican relaciones entre las tablas. Considere el ejemplo de la tabla *Enrollment* (tabla 3.3), en la cual cada renglón representa a un estudiante inscrito en alguno de los cursos que se ofrecen. Los valores de la columna *StdSSN* de la tabla *Enrollment* coinciden con los valores de *StdSSN* de la tabla *Student* (tabla 3.1). Por ejemplo, el primer y tercer renglones de la tabla *Enrollment* tienen el mismo valor en *StdSSN* (123-45-6789) que el primer renglón de la

<sup>2</sup> Los tipos de datos no son estándares entre los DBMS relacionales. Los tipos de datos utilizados en este capítulo son los especificados en el último estándar de SQL. La mayoría de los DBMS soportan estos tipos de datos, aunque sus nombres pueden variar.

**TABLA 3.3**  
Ejemplo de tabla de inscripción

OfferNo	StdSSN	EnrGrade
1234	123-45-6789	3.3
1234	234-56-7890	3.5
4321	123-45-6789	3.5
4321	124-56-7890	3.2

**TABLA 3.4** Ejemplo de tabla de cursos ofrecidos

OfferNo	CourseNo	OffTerm	OffYear	OffLocation	OffTime	FacSSN	OffDays
1111	IS320	SUMMER	2006	BLM302	10:30 AM		MW
1234	IS320	FALL	2005	BLM302	10:30 AM	098-76-5432	MW
2222	IS460	SUMMER	2005	BLM412	1:30 PM		TTH
3333	IS320	SPRING	2006	BLM214	8:30 AM	098-76-5432	MW
4321	IS320	FALL	2005	BLM214	3:30 PM	098-76-5432	TTH
4444	IS320	SPRING	2006	BLM302	3:30 PM	543-21-0987	TTH
5678	IS480	SPRING	2006	BLM302	10:30 AM	987-65-4321	MW
5679	IS480	SPRING	2006	BLM412	3:30 PM	876-54-3210	TTH
9876	IS460	SPRING	2006	BLM307	1:30 PM	654-32-1098	TTH

**FIGURA 3.1**  
Valores que coinciden entre las tablas inscripción, cursos ofrecidos y estudiante

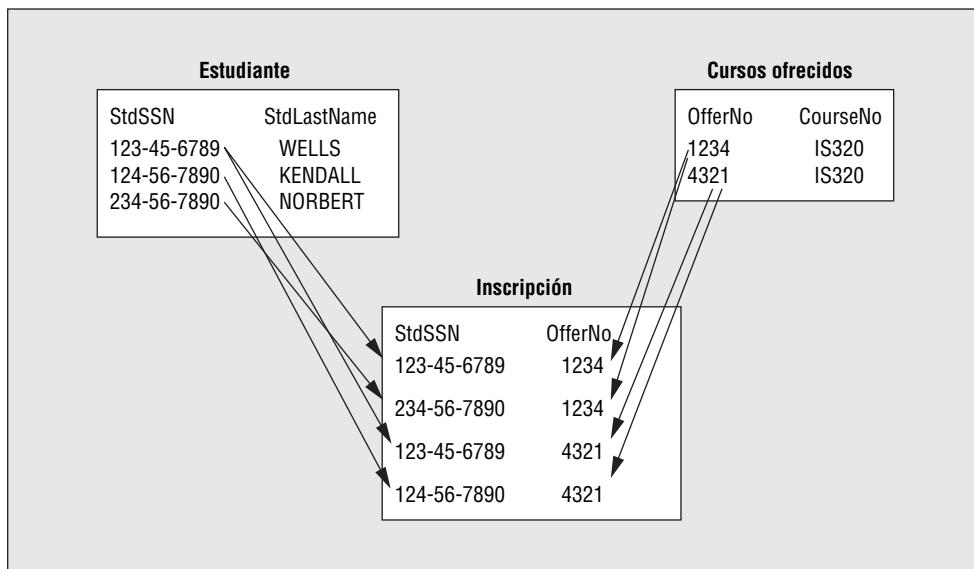


tabla *Student*. De la misma manera, los valores de la columna *OfferNo* de la tabla *Enrollment* coinciden con la columna *OfferNo* de la tabla *Offering* (tabla 3.4). La figura 3.1 muestra una ilustración gráfica de los valores que coinciden.

El concepto de coincidencia de valores es crucial en las bases de datos relacionales. Como observará, las bases de datos por lo general tienen muchas tablas. Incluso una base de datos de tamaño modesto puede tener entre 10 y 15 tablas. Las bases de datos grandes pueden tener cientos de tablas. Para extraer información significativa, por lo general es necesario combinar varias tablas utilizando valores que coincidan. Puede combinar las tablas *Student* y *Enrollment* al hacer que coincidan *Student.StdSSN* y *Enrollment.StdSSN*.<sup>3</sup> De igual forma, al hacer que coincidan *Enrollment.OfferNo* y *Offering.OfferNo*, puede combinar las tablas *Enrollment* y *Offering*. Como observará más adelante en este mismo capítulo, a la operación de combinar tablas con

<sup>3</sup> Cuando las columnas tienen el mismo nombre en dos tablas, se acostumbra anteponer al nombre de la columna el nombre de la tabla y un punto, como en *Student.StdSSN* y *Enrollment.StdSSN*.

**TABLA 3.5**  
**Terminología alterna para bases de datos relacionales**

Orientada a tablas	Orientada a conjuntos	Orientada a registros
Tabla	Relación	Tipo de registro, archivo
Fila	Tuplo	Registro
Columna	Atributo	Campo

valores que coincidan se le conoce como *join* (enlace). Para extraer datos útiles es crucial la comprensión de las conexiones entre las tablas (o de las formas en que se pueden combinar).

### 3.1.3 Terminología alternativa

Usted debe saber que se usa otra terminología además de tabla, fila y columna. La tabla 3.5 muestra tres terminologías más o menos equivalentes. La divergencia de la terminología se debe a los distintos grupos que usan las bases de datos. La terminología orientada a tablas es atractiva para los usuarios finales; la terminología orientada a conjuntos le gusta a los investigadores académicos; y la terminología orientada a registros le gusta a los profesionales de sistemas de información. En la práctica los términos se pueden intercambiar. Por ejemplo, en el mismo enunciado puede ser que se hable tanto de “tablas” como de “campos”. A lo largo de su carrera verá este intercambio de términos.

## 3.2 Reglas de integridad

En la sección anterior aprendió que una base de datos está formada por una colección de tablas interrelacionadas. Para cerciorarse de que una base de datos proporciona información significativa es necesario recurrir a las reglas de integridad. Esta sección describe dos reglas de integridad importantes (integridad de la entidad e integridad referencial), ejemplos de sus usos y una notación para visualizar la integridad referencial.

### 3.2.1 Definición de las reglas de integridad

Integridad de la entidad significa que cada tabla debe tener una columna o combinación de columnas con valores únicos.<sup>4</sup> Único significa que no existen dos filas de una tabla que tengan el mismo valor. Por ejemplo, *StdSSN* en *Student* es única y la combinación de *StdSSN* y *OfferNo* es única en *Enrollment*. La integridad de la entidad asegura que las entidades (personas, cosas y eventos) se identifiquen de forma única en una base de datos. Por razones de auditoría, seguridad y comunicaciones, es importante que las entidades de negocio sean fácilmente rastreables.

Integridad referencial significa que los valores de la columna de una tabla deben coincidir con los valores de la columna de la tabla relacionada. Por ejemplo, el valor *StdSSN* de cada fila de la tabla *Enrollment* debe coincidir con el valor de *StdSSN* de alguna fila de la tabla *Student*. La integridad referencial se cerciora de que una base de datos contenga conexiones válidas. Por ejemplo, es crítico que cada fila de la tabla *Enrollment* contenga un número de seguridad social válido para un estudiante. De lo contrario, algunas inscripciones no tendrán significado, lo que posiblemente genere estudiantes con inscripciones denegadas debido a la inexistencia de los estudiantes que ocuparon esos lugares.

Para una definición más precisa de la integridad de la entidad y de la integridad referencial, se necesitan algunas otras definiciones. A continuación se describen estas definiciones previas y definiciones más precisas.

#### *Definiciones*

- Superllave: Una columna o combinación de columnas que contiene valores únicos para cada renglón. La combinación de todas las columnas de una tabla siempre es una superllave, ya que las filas de una tabla deben ser únicas.<sup>5</sup>

<sup>4</sup> A la integridad de la entidad también se le conoce como integridad de unicidad.

<sup>5</sup> La unicidad de las filas es una característica del modelo relacional que no requiere SQL.

- Llave candidata: Una superllave mínima. Una superllave es mínima si al quitar cualquiera de las columnas ya no es única.
- Valor nulo: Un valor especial que representa la ausencia de un valor presente. Un valor nulo puede indicar que se desconoce el valor presente o que no aplica para una determinada fila.
- Llave primaria: Una llave candidata diseñada de forma especial. La llave primaria de una tabla no puede contener valores nulos.
- Llave foránea: Columna o combinación de columnas en la cual los valores deben coincidir con aquéllos de la llave candidata. Una llave foránea debe tener el mismo tipo de datos que su llave candidata asociada. En la sentencia CREATE TABLE de SQL, una llave foránea debe estar asociada con una llave primaria en lugar de sólo con una llave candidata.

#### *Reglas de integridad*

- Regla de integridad de la entidad: No puede haber dos filas en una tabla que contengan el mismo valor para una llave primaria. Adicionalmente, ninguna fila puede tener un valor nulo para cualquiera de las columnas de una llave primaria.
- Regla de integridad referencial: Solamente se pueden almacenar dos tipos de valores en una llave foránea:
  - un valor que coincide con la llave candidata de alguna fila de la tabla que contenga la llave candidata asociada o
  - un valor nulo.

### 3.2.2 Aplicación de las reglas de integridad

Para ampliar su conocimiento, déjenos aplicar las reglas de integridad a varias tablas de la base de datos de la universidad. La llave primaria de *Student* es *StdSSN*. Una llave primaria se puede designar como parte de la sentencia CREATE TABLE. Para designar *StdSSN* como la llave primaria de *Student*, utilice la cláusula CONSTRAINT para la llave primaria al final de la sentencia CREATE TABLE. El nombre de la restricción (PKStudent) que se encuentra después de la palabra clave CONSTRAINT facilita la identificación de la restricción si ocurre una violación cuando se inserte o actualice una fila.

CREATE TABLE Student

```
(    StdSSN           CHAR(11),
    StdFirstName     VARCHAR(50),
    StdLastName      VARCHAR(50),
    StdCity          VARCHAR(50),
    StdState         CHAR(2),
    StdZip           CHAR(10),
    StdMajor          CHAR(6),
    StdClass          CHAR(2),
    StdGPA            DECIMAL(3,2),
CONSTRAINT PKStudent PRIMARY KEY (StdSSN) )
```

Los números de seguridad social son asignados por el gobierno federal; por lo tanto, la universidad no tiene que asignarlos. En otros casos, los valores principales son asignados por una organización. Por ejemplo, números de cliente, números de producto y números de empleado generalmente son asignados por la organización que controla la base de datos en cuestión. En estos casos, se requiere la generación automática de valores únicos. Algunos DBMS soportan la generación automática de valores únicos, tal como se explica en el apéndice 3.C.

#### *Variaciones de la integridad de la entidad*

Las llaves candidatas que no son llaves primarias se declaran con la palabra clave UNIQUE. La tabla *Course* (vea tabla 3.6) contiene dos llaves candidatas: *CourseNo* (llave primaria) y *CrsDesc*

**TABLA 3.6**  
**Ejemplo de tabla de**  
**cursos**

CourseNo	CrsDesc	CrsUnits
IS320	FUNDAMENTOS DE NEGOCIO	4
IS460	ANÁLISIS DE SISTEMAS	4
IS470	COMUNICACIONES DE DATOS DE NEGOCIOS	4
IS480	FUNDAMENTOS DE BASE DE DATOS	4

(descripción del curso). La columna *CourseNo* es la llave primaria, ya que es más estable que la columna *CrsDesc*. Las descripciones del curso pueden variar con el tiempo, pero los números del curso se conservan.

```
CREATE TABLE Course
(
    CourseNo           CHAR(6),
    CrsDesc            VARCHAR(250),
    CrsUnits           SMALLINT,
    CONSTRAINT PKCourse PRIMARY KEY(CourseNo),
    CONSTRAINT UniqueCrsDesc UNIQUE (CrsDesc) )
```

Algunas tablas necesitan más de una columna en la llave primaria. En la tabla *Enrollment*, la combinación de *StdSSN* y *OfferNo* es la única llave candidata. Se necesitan ambas columnas para identificar una fila. A una llave primaria formada por más de una columna se le conoce como llave primaria compuesta o combinada.

```
CREATE TABLE Enrollment
(
    OfferNo            INTEGER,
    StdSSN             CHAR(11),
    EnrGrade           DECIMAL(3,2),
    CONSTRAINT PKEnrollment PRIMARY KEY(OfferNo, StdSSN) )
```

Las superllaves que no son mínimas no son importantes, ya que son comunes y contienen columnas que no contribuyen a la propiedad de unicidad. Por ejemplo, la combinación de *StdSSN* y *StdLastName* es única. Sin embargo, si se elimina *StdLastName*, *StdSSN* sigue siendo única.

#### *Integridad referencial*

Para la integridad referencial, las columnas *StdSSN* y *OfferNo* son llaves foráneas en la tabla *Enrollment*. La columna *StdSSN* hace referencia a la tabla *Student* y la columna *OfferNo* hace referencia a la tabla *Offering* (tabla 3.4). Una fila de *Offering* representa un curso dado en un periodo académico (verano, invierno, etc.), año, fecha, lugar y días de la semana. La llave primaria de *Offering* es *OfferNo*. Un curso como el IS480 tendrá diferente número de oferta cada vez que se ofrezca.

Las restricciones de la integridad referencial se pueden definir de forma similar a como se definen las llaves primarias. Por ejemplo, para definir las llaves foráneas de *Enrollment*, debe utilizar las cláusulas **CONSTRAINT** para las llaves foráneas al final de la sentencia **CREATE TABLE**, como se muestra en la revisión de la sentencia **CREATE TABLE** para la tabla *Enrollment*.

```
CREATE TABLE Enrollment
(
    OfferNo            INTEGER,
    StdSSN             CHAR(11),
    EnrGrade           DECIMAL(3,2),
    CONSTRAINT PKEnrollment PRIMARY KEY(OfferNo, StdSSN),
    CONSTRAINT FKOfferNo FOREIGN KEY (OfferNo) REFERENCES Offering,
    CONSTRAINT FKStdSSN FOREIGN KEY (StdSSN) REFERENCES Student )
```

Aunque la integridad referencial permite que las llaves foráneas tengan valores nulos, no es común que los tengan. Cuando una llave foránea es parte de una llave primaria, no se permiten los valores nulos gracias a la regla de integridad de la entidad. Por ejemplo, no se permiten valores nulos para *Enrollment.StdSSN* ni para *Enrollment.OfferNo*, ya que cada una de las columnas es parte de la llave primaria.

Cuando una llave foránea no es parte de una primaria su uso dicta si se permiten los valores nulos. Por ejemplo, *Offering.CourseNo*, una llave foránea que hace referencia a *Course* (tabla 3.4), no es parte de una llave primaria y aun así no permite valores nulos. En la mayoría de las universidades no se puede ofrecer un curso antes de su aprobación. Por lo tanto, una oferta no se puede insertar sin un curso relacionado.

Las palabras clave NOT NULL indican que una columna no puede tener valores nulos, tal como se muestran en la sentencia CREATE TABLE de la tabla *Offering*. Las restricciones NOT NULL son restricciones internas asociadas con una columna en específico. En contraste, las restricciones para las llaves primaria y foránea de la sentencia CREATE TABLE de la tabla *Offering* son restricciones en las que las columnas asociadas se deben especificar dentro de una restricción. Los nombres de las restricciones se deben usar tanto en la tabla como en las restricciones internas para facilitar la identificación cuando ocurra una violación.

```
CREATE TABLE Offering
(
    OfferNo          INTEGER,
    CourseNo         CHAR(6)      CONSTRAINT OffCourseNoRequired NOT NULL,
    OffLocation      VARCHAR(50),
    OffDays          CHAR(6),
    OffTerm          CHAR(6)      CONSTRAINT OffTermRequired NOT NULL,
    OffYear          INTEGER     CONSTRAINT OffYearRequired NOT NULL,
    FacSSN           CHAR(11),
    OffTime          DATE,
    CONSTRAINT PKOffering PRIMARY KEY (OfferNo),
    CONSTRAINT FKCourseNo FOREIGN KEY(CourseNo) REFERENCES Course,
    CONSTRAINT FKFacSSN FOREIGN KEY(FacSSN) REFERENCES Faculty
)
```

**relación autorreferencial**  
una relación en la cual una llave foránea hace referencia a la misma tabla. Las relaciones que hacen referencia a sí mismas representan asociaciones entre miembros del mismo conjunto.

En contraste, *Offering.FacSSN* puede ser nulo en referencia al miembro facultativo que enseña el curso ofertado. La tabla *Faculty* (tabla 3.7) almacena datos acerca de los instructores de los cursos. Un valor nulo para *Offering.FacSSN* significa que aún no se ha asignado a un facultativo para que enseñe el curso ofertado. Por ejemplo, no se ha asignado instructor en la primera y tercera filas de la tabla 3.4. Como las ofertas de cursos se deben programar incluso con un año de anticipación, es probable que no se conozca a los instructores de algunos cursos ofertados sino hasta después de haber almacenado la fila del curso ofrecido. Por lo mismo, es prudente permitir valores nulos en la tabla *Offering*.

#### *Integridad referencial para las relaciones autorreferenciales (unitarias)*

Una restricción de integridad referencial que incluye una tabla sencilla se conoce como relación autorreferencial o unitaria. Las relaciones autorreferenciales no son comunes, pero cuando ocurren son importantes. En la base de datos de la universidad, un miembro del facultativo puede supervisar a otros miembros de su facultad y a la vez ser supervisado por un miembro de la facultad. Por ejemplo, Victoria Emmanuel (segunda fila) supervisa a Leonard Fibon (tercera fila).

**TABLA 3.7 Ejemplo de tabla de facultativos**

FacSSN	FacFirstName	FacLastName	FacCity	FacState	FacDept	FacRank	FacSalary	FacSupervisor	FacHireDate	FacZipCode
098-76-5432	LEONARD	VINCE	SEATTLE	WA	MS	ASST	\$35,000	654-32-1098	01-Apr-95	98111-9921
543-21-0987	VICTORIA	EMMANUEL	BOTHELL	WA	MS	PROF	\$120,000		01-Apr-96	98011-2242
654-32-1098	LEONARD	FIBON	SEATTLE	WA	MS	ASSC	\$70,000	543-21-0987	01-Apr-95	98121-0094
765-43-2109	NICKI	MACON	BELLEVUE	WA	FIN	PROF	\$65,000		01-Apr-97	98015-9945
876-54-3210	CRISTOPHER	COLAN	SEATTLE	WA	MS	ASST	\$40,000	654-32-1098	01-Apr-99	98114-1332
987-65-4321	JULIA	MILLS	SEATTLE	WA	FIN	ASSC	\$75,000	765-43-2109	01-Apr-00	98114-9954

La columna *FacSupervisor* muestra esta relación: el valor de *FacSupervisor* de la tercera fila (543-21-0987) coincide con el valor de *FacSSN* de la segunda fila. Una restricción de integridad referencial que involucra a la columna *FacSupervisor* representa la relación autorreferencial. En la sentencia CREATE TABLE, la restricción de integridad referencial para una relación autorreferencial se puede escribir de la misma forma que el resto de las restricciones de integridad referencial.

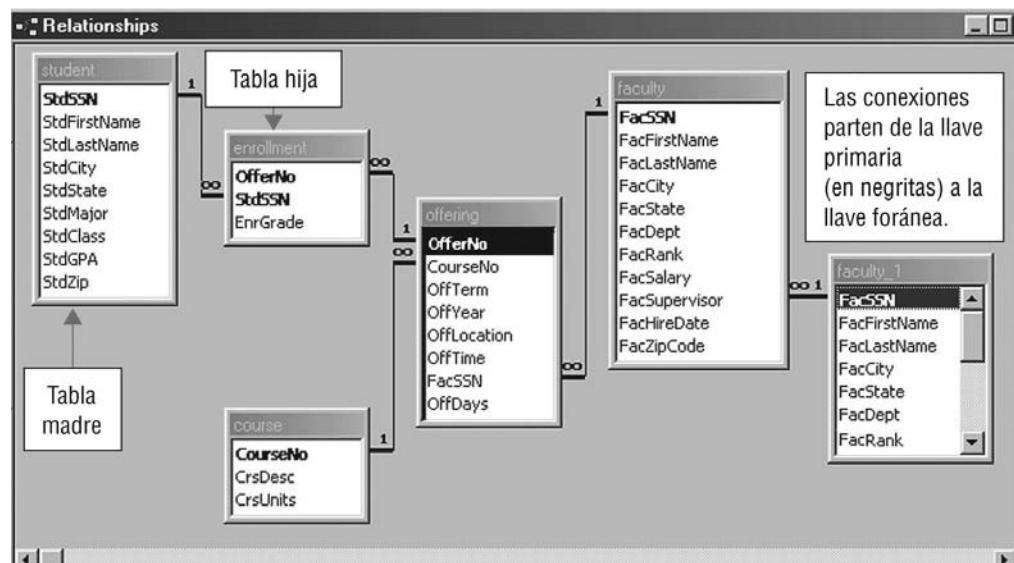
```
CREATE TABLE Faculty
(
    FacSSN           CHAR(11),
    FacFirstName     VARCHAR(50)   CONSTRAINT FacFirstNameRequired NOT NULL,
    FacLastName      VARCHAR(50)   CONSTRAINT FacLastNameRequired NOT NULL,
    FacCity          VARCHAR(50)   CONSTRAINT FacCityRequired NOT NULL,
    FacState         CHAR(2)       CONSTRAINT FacStateRequired NOT NULL,
    FacZipCode        CHAR(10)      CONSTRAINT FacZipCodeRequired NOT NULL,
    FacHireDate      DATE,
    FacDept          CHAR(6),
    FacRank          CHAR(4),
    FacSalary         DECIMAL(10,2),
    FacSupervisor    CHAR(11),
    CONSTRAINT PKFaculty PRIMARY KEY (FacSSN),
    CONSTRAINT FKFacSupervisor FOREIGN KEY (FacSupervisor) REFERENCES Faculty
)
```

### 3.2.3 Representación gráfica de la integridad referencial

En años recientes, los DBMS comerciales han proporcionado representaciones gráficas para las restricciones de integridad referencial. La representación gráfica hace que la integridad referencial sea más fácil de definir y comprender que con la representación textual de la sentencia CREATE TABLE. Además, una representación gráfica soporta el acceso no procedural a datos.

Para ilustrar una representación gráfica, estudiemos la ventana de relaciones de Microsoft Access. Access proporciona la ventana de relaciones para definir y desplegar de forma visual las restricciones de la integridad referencial. La figura 3.2 muestra la ventana de relaciones para las tablas de la base de datos de la universidad. Cada línea representa una restricción de integridad referencial o relación. En una relación, la tabla de llave primaria se conoce como madre o tabla

**FIGURA 3.2**  
Ventana de relaciones para la base de datos de la universidad



“1” (por ejemplo, *Student*) y a la de llave foránea (por ejemplo, *Enrollment*) se le conoce como tabla hija o “M” (muchos).

#### **relación 1-M**

una conexión entre dos tablas en la cual una fila de la tabla madre se puede referenciar con muchas filas de una tabla hija. Las relaciones 1-M son el tipo más común de relaciones.

#### **relación M-N**

una conexión entre dos tablas en la que las filas de cada tabla se pueden relacionar con muchas filas de otra tabla. Las relaciones M-N no se pueden representar de forma directa en el modelo relacional. Dos relaciones 1-M y un enlace o tabla asociativa representan una relación M-N.

A la relación de *Student* a *Enrollment* se le llama “1-M” (de uno a muchos), ya que un estudiante se puede relacionar con muchas inscripciones, pero una inscripción sólo se puede relacionar con un estudiante. De manera similar, la relación de la tabla *Offering* a la tabla *Enrollment* significa que un curso que se ofrece puede relacionarse con muchas inscripciones, pero una inscripción sólo puede relacionarse con un curso. Usted debe practicar escribiendo enunciados similares para las otras relaciones de la figura 3.2.

Las relaciones M-N (muchos a muchos) no se representan de forma directa en el modelo relacional. Una relación M-N significa que las filas de cada tabla se pueden relacionar con muchas filas de otra tabla. Por ejemplo, un estudiante se inscribe en muchos cursos y un curso contiene muchos estudiantes. En el modelo relacional, una pareja de relaciones 1-M y un enlace o tabla asociativa representan una relación M-N. En la figura 3.2, la tabla de enlaces *Enrollment* y sus relaciones con *Offering* y *Student* representan una relación M-N entre las tablas *Student* y *Offering*.

Las relaciones autorreferenciales se representan de forma indirecta en la ventana relaciones. La relación autorreferencial que involucra a *Faculty* se representa como una relación entre las tablas *Faculty* y *Faculty\_1*. *Faculty\_1* no es una tabla real y solamente está creada dentro de la ventana de relaciones de Access. Access sólo puede mostrar de forma indirecta las relaciones autorreferenciales.

Una representación gráfica como la ventana relaciones facilita la identificación de las tablas que deben combinarse para responder a una solicitud de extracción. Por ejemplo, suponga que usted quiere encontrar los instructores que imparten cursos con la palabra “base de datos” dentro de la descripción del curso. Es evidente que necesitará la tabla *Course* para encontrar los cursos de “base de datos”. También necesitará la tabla *Faculty* para desplegar los datos del instructor. La figura 3.2 muestra que también necesita la tabla *Offering*, ya que *Course* y *Faculty* no están conectadas de forma directa. En su lugar, *Course* y *Faculty* se conectan a través de *Offering*. Por lo tanto, la observación de las relaciones le ayuda a identificar las tablas que necesita para completar las solicitudes de extracción. Antes de intentar los problemas de extracción de los capítulos siguientes, debe estudiar cuidadosamente una representación gráfica de las relaciones. Debe construir su propio diagrama en caso de que no haya uno disponible.

### 3.3 Acciones para eliminar y actualizar filas referenciadas

Para cada restricción de una integridad referencial, debe considerar cuidadosamente las acciones sobre las filas referenciadas en las tablas madre de las relaciones 1-M. Una fila madre está referenciada si hay filas en la tabla hija con idénticos valores de la llave foránea al valor de la llave primaria de la fila de la tabla madre. Por ejemplo, la primera fila de la tabla *Course* (tabla 3.6) con *CourseNo* “IS320” está referenciada por la primera fila de la tabla *Offering* (tabla 3.4). Es natural considerar lo que le sucede a las filas relacionadas de *Offering* cuando el renglón referenciado de *Course* se elimina o el *CourseNo* se actualiza. De forma general, estas inquietudes se establecen como

**Eliminación de una fila referenciada:** ¿Qué le pasa a las filas relacionadas (es decir, filas de la tabla hija con idéntico valor de llave foránea) cuando se elimina la fila referenciada de la tabla madre?

**Actualización de la llave primaria de una fila referenciada:** ¿Qué le pasa a las filas relacionadas cuando se actualiza la llave primaria de la fila referenciada en la tabla madre?

Las acciones sobre las filas referenciadas son importantes cuando se hacen cambios a las filas de una base de datos. Cuando se desarrollan formularios de captura de datos (descritos en el capítulo 10), las acciones sobre las filas referenciadas pueden ser de mucha importancia. Por ejemplo, si un formulario de captura de datos permite la eliminación de renglones de la tabla *Course*, las acciones sobre las filas relacionadas de la tabla *Offering* se deben planear de forma cuidadosa. De lo contrario, la base de datos puede volverse inconsistente o difícil de usar.

### Acciones posibles

Existen varias acciones posibles para dar respuesta a la eliminación de una fila referenciada o a la actualización de la llave primaria de una fila referenciada. La acción apropiada depende de las tablas involucradas. La siguiente lista describe las acciones y proporciona ejemplos de su uso.

- **Restringir:**<sup>6</sup> No permitir la acción en la fila referenciada. Por ejemplo, no permitir que una fila de *Student* se borre si existen renglones relacionados en *Enrollment*. De forma similar, no permitir que *Student.StdSSN* se modifique si existen filas en *Enrollment*.
- **Cascada:** Realizar la misma acción (acción en cascada) en las filas relacionadas. Por ejemplo, si se elimina un *Student*, se deben eliminar las filas relacionadas en *Enrollment*. De forma similar, si *Student.StdSSN* se modifica en alguna fila, se debe actualizar *StdSSN* en las filas relacionadas de *Enrollment*.
- **Nulificar:** Igualar a nulo la llave foránea de las filas relacionadas. Por ejemplo, si se elimina una fila de *Faculty*, entonces igualar *FacSSN* a NULL en las filas relacionadas de *Offering*. De forma similar, si se actualiza *Faculty.FacSSN*, entonces igualar *FacSSN* a NULL en las filas relacionadas de *Offering*. La acción nulificar no se permite cuando la llave foránea no permite valores nulos. Por ejemplo, la opción nulificar no es válida cuando se eliminan renglones de la tabla *Student*, ya que *Enrollment.StdSSN* es parte de la llave primaria.
- **Por omisión:** Igualar la llave foránea de las filas relacionadas al valor por omisión. Por ejemplo, si se elimina una fila de *Faculty*, entonces se debe igualar *FacSSN* al valor por omisión del facultativo en relación con las filas *Offering*. El valor por omisión del facultativo puede interpretarse como “se anunciará”. De forma similar, si se actualiza *Factulty.FacSSN*, igualar *FacSSN* a nulo en las filas relacionadas de *Offering*. La acción por omisión es una alternativa a la acción de nulificar, ya que la acción por omisión evita valores nulos.

Las acciones para eliminar y actualizar se pueden especificar en SQL utilizando las cláusulas ON DELETE y ON UPDATE. Estas cláusulas son parte de restricciones de las llaves foráneas. Por ejemplo, la declaración revisada CREATE TABLE para la tabla *Enrollment* muestra las cláusulas ON DELETE y ON UPDATE. Las palabras claves CASCADE, SET NULL y SET DEFAULT pueden utilizarse para especificar de la segunda a la cuarta opción, respectivamente.

```
CREATE TABLE Enrollment
(
    OfferNo          INTEGER,
    StdSSN           CHAR(11),
    EnrGrade         DECIMAL(3,2),
    CONSTRAINT PKEnrollment PRIMARY KEY (OfferNo, StdSSN),
    CONSTRAINT FKOfferNo FOREIGN KEY (OfferNo) REFERENCES Offering
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
    CONSTRAINT FKStdSSN FOREIGN KEY (StdSSN) REFERENCES Student
        ON DELETE RESTRICT
        ON UPDATE CASCADE
)
```

Antes de terminar esta sección, debe comprender el impacto de las filas referenciadas en las operaciones de inserción. Se debe insertar una fila referenciada antes de sus filas relacionadas. Por ejemplo, antes de insertar una fila en la tabla *Enrollment*, deben existir las filas referenciadas de las tablas *Student* y *Offering*. La integridad referencial indica el orden de inserción de las filas de las distintas tablas. Cuando se diseñan formularios de captura de datos, debe considerar cuidadosamente el impacto de la integridad referencial en el orden en el que los usuarios llenan los formularios.

<sup>6</sup> Existe una acción relacionada definida por las palabras clave NO ACTION. La diferencia entre RESTRICT y NO ACTION involucra el concepto de restricciones delegadas de integridad, comentadas en el capítulo 15.

## 3.4 Operadores de álgebra relacional

En las secciones previas de este capítulo, usted estudió la terminología y las reglas de integridad de las bases de datos relacionales con el objetivo de entender su existencia. En lo particular, se puso énfasis en la comprensión de las conexiones entre las tablas como un prerequisito para extraer información útil. Esta sección describe algunos operadores básicos que se pueden usar para extraer información útil de una base de datos relacional.

Puede pensar el álgebra relacional de manera similar al álgebra numérica, excepto porque los objetos son diferentes: el álgebra se aplica a los números y el álgebra relacional se aplica a las tablas. En álgebra, cada operador transforma uno o más números en otro número. De forma similar, cada operador de álgebra relacional transforma una tabla (o dos tablas) en una tabla nueva.

Esta sección pone énfasis en el estudio por separado de cada operador de álgebra relacional. Debe comprender el propósito y la entrada de cada uno de los operadores. Aunque es posible combinar operadores para hacer fórmulas complicadas, este nivel de comprensión no es importante para desarrollar las habilidades de formulación de consultas. El uso de álgebra relacional por sí mismo puede ser inconveniente para escribir consultas, debido a detalles como el orden de las operaciones y los paréntesis. Por lo mismo, únicamente debe comprender el significado de cada operador y no cómo combinarlos para escribir expresiones. El álgebra relacional agrupa a los operadores en tres categorías. Primero se presentan los operadores más usados (restricción, proyección y enlace). También se presenta el operador del producto cruz extendido para proporcionar los antecedentes del operador de enlace (*join*). El conocimiento de estos operadores le ayudará a formular un gran porcentaje de las consultas. Los operadores más especializados se tratan en las últimas partes de la sección. Los operadores más especializados incluyen el conjunto de operadores tradicionales (unión, intersección y diferencia) y los operadores avanzados (resumir y dividir). El conocimiento de estos operadores le ayudará a formular las consultas más difíciles.

### 3.4.1 Operadores de restricción (*select*) y proyección (*project*)

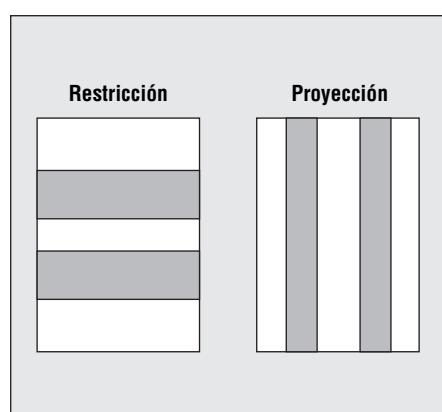
Los operadores de restricción (también conocidos como *select*) y proyección generan subconjuntos de una tabla.<sup>7</sup> Estos operadores se usan ampliamente debido a que los usuarios por lo común desean observar un subconjunto en vez de una tabla completa. Estos operadores también son populares, ya que son fáciles de entender.

Los operadores de restricción y proyección generan una tabla de salida, que es un subconjunto de una tabla de entrada (figura 3.3). La restricción genera un subconjunto de renglones, mientras que la proyección genera un subconjunto de columnas. La restricción usa una condición o expresión lógica para indicar que las filas se deben conservar en la salida. La proyección usa

#### restricción

un operador que extrae un subconjunto de filas de la tabla de entrada satisfaciendo una condición determinada.

**FIGURA 3.3**  
Representación gráfica de los operadores de restricción y proyección



<sup>7</sup> En este libro se usa el nombre de operador restricción para evitar la confusión con la sentencia SELECT de SQL. Al operador se le conoce más como select.

**TABLA 3.8 Resultado de la operación de restricción de la tabla de ejemplo *Offering* (tabla 3.4)**

OfferNo	CourseNo	OffTerm	OffYear	OffLocation	OffTime	FacSSN	OffDays
3333	IS320	SPRING	2006	BLM214	8:30 AM	098-76-5432	MW
5678	IS480	SPRING	2006	BLM302	10:30 AM	987-65-4321	MW

**TABLA 3.9 Resultado de una operación de proyección en *Offering.CourseNo***

CourseNo
IS320
IS460
IS480

**proyección**

un operador que extrae un subconjunto específico de columnas de la tabla de entrada.

una lista de nombres de columna para indicar las columnas que deben conservarse en la salida. Los operadores restricción y proyección se usan generalmente juntos, ya que las tablas pueden tener filas y columnas. Es raro el caso en que un usuario quiera ver todas las filas y las columnas.

La expresión lógica que se usa en el operador restricción puede incluir comparaciones que incluyen columnas y constantes. Las expresiones lógicas complejas se pueden formar utilizando los operadores lógicos AND, OR y NOT. Por ejemplo, la tabla 3.8 muestra el resultado de una operación de restricción hecha sobre la tabla 3.4, donde la expresión lógica es: OffDays = 'MW' AND OffTerm = 'SPRING' AND OffYear = 2006.

Una operación de proyección puede tener efectos secundarios. Hay ocasiones en que después de extraer un subconjunto de columnas, hay renglones duplicados. Cuando esto ocurre, el operador de proyección elimina las filas duplicadas. Por ejemplo, si *Offering.CourseNo* es la única columna que se usa en la operación de proyección, sólo se encuentran tres filas en el resultado (tabla 3.9), aun cuando la tabla *Offering* (tabla 3.4) tenga nueve filas. La columna *Offering.CourseNo* contiene sólo tres valores únicos en la tabla 3.4. Tenga en cuenta que si la llave primaria o la llave candidata se incluye en la lista de las columnas, la tabla resultante no tendrá duplicados. Por ejemplo, si se incluyó *OfferNo* en la lista de columnas, la tabla resultante tendrá nueve renglones sin necesidad de eliminar los duplicados.

Este efecto secundario se debe a la naturaleza matemática del álgebra relacional. En el álgebra relacional, a las tablas se les considera como conjuntos. Debido a que los conjuntos no tienen duplicados, la eliminación de duplicados es un posible efecto secundario del operador de proyección. Los lenguajes comerciales como SQL usualmente toman un punto de vista más pragmático. Debido a que la eliminación de duplicados puede ser costosa en términos de cómputo, los duplicados no se eliminan a menos que el usuario lo solicite de forma específica.

### 3.4.2 Operador de producto cruz extendido

El operador de producto cruz extendido (*extended cross product*) puede combinar dos tablas cualesquiera. Otros operadores para combinar tablas tienen condiciones sobre las tablas a combinar. Debido a su naturaleza sin restricciones, el operador de producto cruz extendido puede generar tablas con datos excesivos. Este operador es importante porque es un cimiento para el operador de enlace (*join*). Puede ser útil el conocimiento del operador de producto cruz extendido cuando se inicia en el aprendizaje del operador de enlace. Después de obtener experiencia con el operador de enlace, no tendrá que referirse al operador de producto cruz extendido.

El operador de producto cruz extendido (producto, para abreviarlo) muestra todas las combinaciones posibles de dos tablas.<sup>8</sup> El producto de dos tablas es una nueva tabla formada por todas las combinaciones posibles de las filas de las dos tablas de entrada. La figura 3.4 ilustra un producto de dos tablas de una sola columna. Cada fila resultante está formada por las columnas de la tabla *Faculty* (sólo *FacSSN*) y las columnas de la tabla *Student* (sólo *StdSSN*). El nombre del operador (producto) se deriva a partir del número de filas en el resultado. El número de filas

**producto cruz extendido**

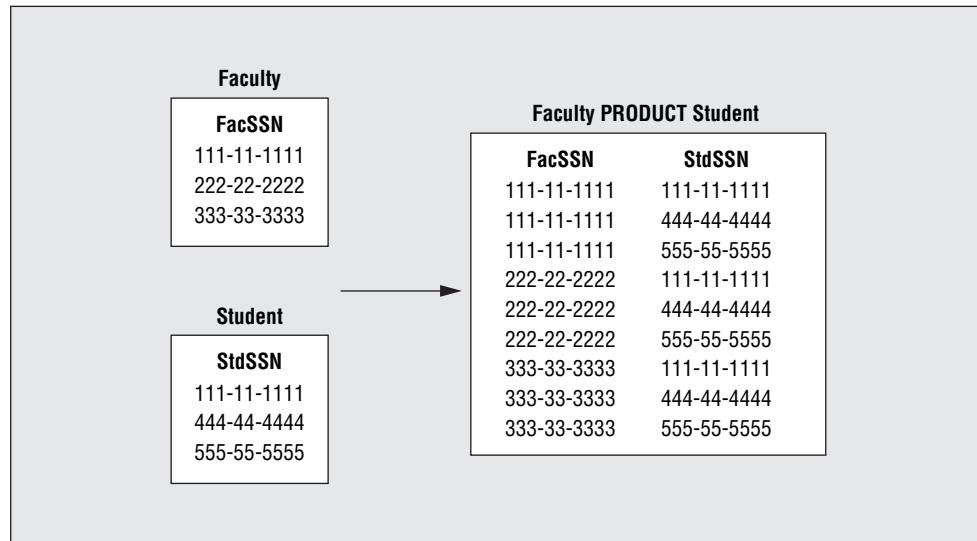
un operador que construye una tabla formada por todas las combinaciones de filas de cada una de las dos tablas de entrada.

<sup>8</sup> Al operador de producto cruz extendido también se le conoce como producto cartesiano, en reconocimiento al matemático francés René Descartes.

de la tabla resultante es el producto del número de filas de las dos tablas de entrada. En contraste, el número de columnas resultantes es la suma de las columnas de las dos tablas de entrada. En la figura 3.4, la tabla resultante tiene nueve filas y dos columnas.

En otro ejemplo, considere el producto de las tablas de ejemplo *Student* (tabla 3.10) y *Enrollment* (tabla 3.11). La tabla resultante (tabla 3.12) tiene nueve filas ( $3 \times 3$ ) y siete columnas ( $4 + 3$ ). Tenga en cuenta que la mayoría de las filas del resultado carecen de significado, ya que sólo tres filas tienen el mismo valor para *StdSSN*.

**FIGURA 3.4**  
Ejemplo de producto cruz



**TABLA 3.10**  
Tabla de ejemplo de *Student*

StdSSN	StdLastName	StdMajor	StdClass
123-45-6789	WELLS	IS	FR
124-56-7890	NORBERT	FIN	JR
234-56-7890	KENDALL	ACCT	JR

**TABLA 3.11**  
Tabla de ejemplo de *Enrollment*

OfferNo	StdSSN	EnrGrade
1234	123-45-6789	3.3
1234	234-56-7890	3.5
4321	124-56-7890	3.2

**TABLA 3.12** PRODUCTO entre *Student* y *Enrollment*

Student.StdSSN	StdLastName	StdMajor	StdClass	OfferNo	Enrollment.StdSSN	EnrGrade
123-45-6789	WELLS	IS	FR	1234	123-45-6789	3.3
123-45-6789	WELLS	IS	FR	1234	234-56-7890	3.5
123-45-6789	WELLS	IS	FR	4321	124-56-7890	3.2
124-56-7890	NORBERT	FIN	JR	1234	123-45-6789	3.3
124-56-7890	NORBERT	FIN	JR	1234	234-56-7890	3.5
124-56-7890	NORBERT	FIN	JR	4321	124-56-7890	3.2
234-56-7890	KENDALL	ACCT	JR	1234	123-45-6789	3.3
234-56-7890	KENDALL	ACCT	JR	1234	234-56-7890	3.5
234-56-7890	KENDALL	ACCT	JR	4321	124-56-7890	3.2

Como lo muestran estos ejemplos, el operador de producto cruz extendido usualmente genera datos excesivos. El exceso de datos es tan malo como la carencia de ellos. Por ejemplo, el producto de una tabla de estudiantes de 30 000 filas y de la tabla de inscripciones de 300 000 filas es una tabla de ¡9 000 millones de filas! Muchas de estas filas serían combinaciones sin sentido. Así que es raro que se necesite por sí misma una operación de producto cruz. En su lugar, la importancia del operador de producto cruz se debe a que es un cimiento para otros operadores, tal como el operador de enlace.

### 3.4.3 Operador de enlace (*join*)

El enlace (*join*) es el operador más usado para combinar tablas. La combinación de tablas es importante debido a que la mayoría de bases de datos tienen muchas tablas. El enlace difiere del producto cruz porque requiere de una condición de coincidencia sobre las filas de dos tablas. La mayoría de las tablas se combinan de esta forma. En gran medida, su habilidad para extraer datos útiles dependerá de su habilidad al usar el operador de enlace.

El operador enlace construye una tabla nueva al combinar filas de dos tablas que coinciden con una condición de enlace. Comúnmente, la condición de enlace especifica que dos filas tienen un valor idéntico en una o más columnas. Cuando la condición de enlace involucra igualdad, al enlace se le conoce como equienlace (*equi-join*), dada la igualdad. La figura 3.5 muestra un enlace de ejemplo entre las tablas *Faculty* y *Offering* en donde la condición de enlace es que las columnas *FacSSN* sean iguales. Observe que sólo se muestran unas cuantas columnas para simplificar la ilustración. Las flechas indican cómo es que las filas de las tablas iniciales se combinan para formar las filas de la tabla resultante. Por ejemplo, la primera columna de la tabla *Faculty* se combina con la primera y tercera filas de la tabla *Offering* para generar dos filas en la tabla resultante.

La operación de enlace más común es la del operador enlace natural (*natural join*). En una operación de enlace natural, la condición de enlace es de igualdad (equienlace), una de las columnas de enlace se elimina y las columnas de enlace tienen el mismo nombre no limitado.<sup>9</sup> En la figura 3.5 la tabla resultante contiene solamente tres columnas, ya que el enlace natural elimina una de las columnas *FacSSN*. La columna particular eliminada no importa (*Faculty.FacSSN* u *Offering.FacSSN*).

#### **enlace (*join*)**

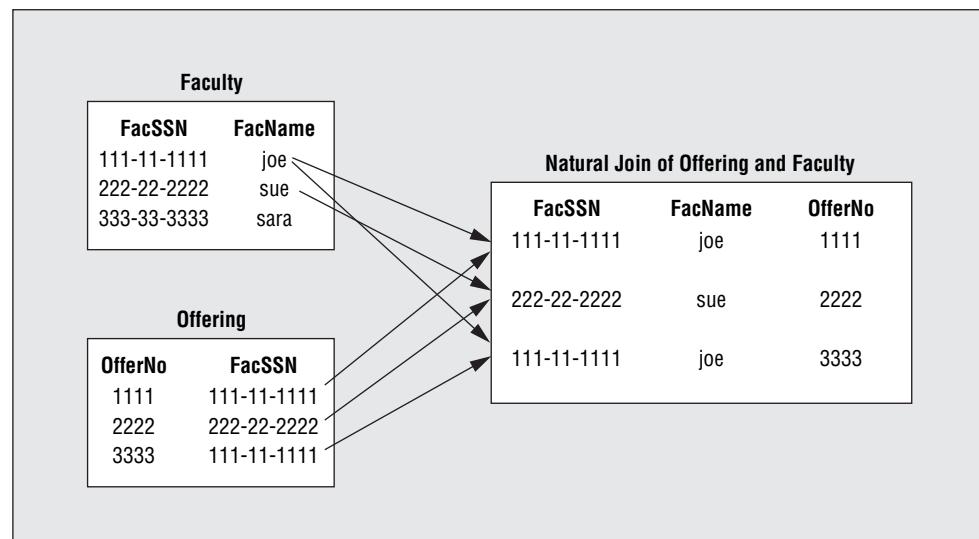
un operador que genera una tabla que contiene las filas que coinciden con una condición e involucra una columna de cada tabla inicial.

#### **enlace natural**

##### **(*natural join*)**

un operador de enlace muy común en donde la condición de coincidencia es la igualdad (equienlace), una de las columnas que coinciden se elimina de la tabla resultante y las columnas que coinciden tienen los mismos nombres no limitados.

**FIGURA 3.5**  
Ejemplo de operación de enlace natural



<sup>9</sup> Un nombre no limitado es el nombre de la columna sin el nombre de la tabla. El nombre completo de una columna incluye el nombre de la tabla. Por lo tanto, los nombres completos de las columnas de enlace de la figura 3.5 son *Faculty.FacSSN* y *Offering.FacSSN*.

**TABLA 3.13**  
**Tabla de ejemplo de**  
***Student***

StdSSN	StdLastName	StdMajor	StdClass
123-45-6789	WELLS	IS	FR
124-56-7890	NORBERT	FIN	JR
234-56-7890	KENDALL	ACCT	JR

**TABLA 3.14**  
**Tabla de ejemplo de**  
***Enrollment***

OfferNo	StdSSN	EnrGrade
1234	123-45-6789	3.3
1234	234-56-7890	3.5
4321	124-56-7890	3.2

**TABLA 3.15**  
**Enlace natural de**  
***Student* y *Enrollment***

Student.StdSSN	StdLastName	StdMajor	StdClass	OfferNo	EnrGrade
123-45-6789	WELLS	IS	FR	1234	3.3
124-56-7890	NORBERT	FIN	JR	4321	3.2
234-56-7890	KENDALL	ACCT	JR	1234	3.5

Como otro ejemplo, considere el enlace natural de *Student* (tabla 3.13) y *Enrollment* (tabla 3.14) que se muestra en la tabla 3.15. En cada fila del resultado, *Student.StdSSN* coincide con *Enrollment.StdSSN*. Sólo una de las columnas de enlace se incluye en el resultado. Se muestra *Student.StdSSN* de forma arbitraria, aunque se puede incluir *Enrollment.StdSSN* sin modificar el resultado.

#### *Derivación del enlace natural*

El operador de enlace natural no es primitivo porque se puede deducir de otros operadores. El operador de enlace natural consta de tres pasos:

1. El producto de la operación de combinar las filas.
2. Una operación de restricción para eliminar las filas que no satisfacen la condición del enlace.
3. Una operación de proyección para eliminar una de las columnas del enlace.

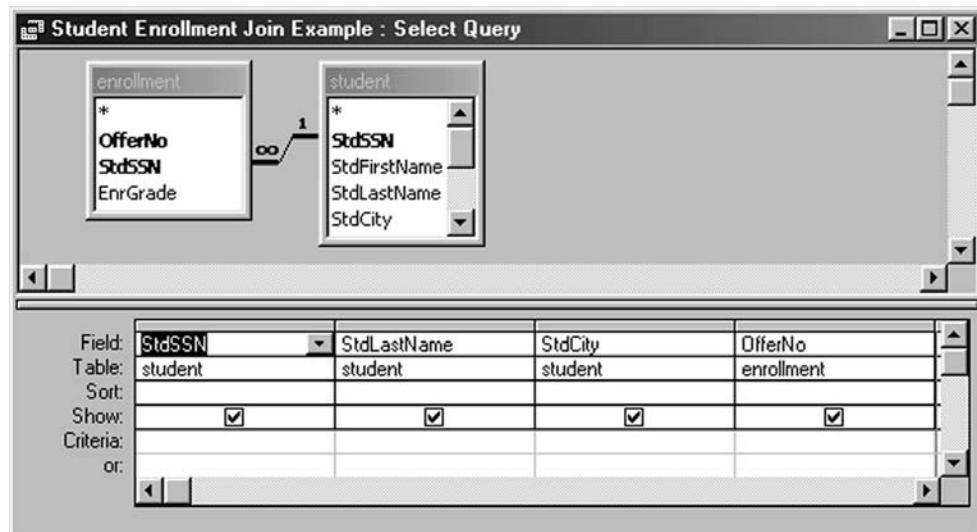
Ilustremos estos pasos; el primero para generar un enlace natural de la tabla 3.15 es el resultado del producto que se muestra en la tabla 3.12. El segundo paso es conservar sólo las filas que coincidan (filas 1, 6 y 8 de la tabla 3.12). Se usa una operación de restricción con *Student.StdSSN* = *Enrollment.StdSSN* como condición de restricción. El último paso es eliminar una de las columnas del enlace (*Enrollment.StdSSN*). La operación de proyección incluye todas las columnas, excepto *Enrollment.StdSSN*.

Aunque el operador de enlace no es primitivo, se puede conceptualizar de forma directa sin sus operaciones primitivas. Cuando usted inicie el aprendizaje del operador de enlace quizás le sea útil deducir los resultados utilizando las operaciones implícitas. Como ejercicio se recomienda que deduzca el resultado de la figura 3.5. Después de aprender sobre el operador de enlace no necesitará más utilizar las operaciones implícitas.

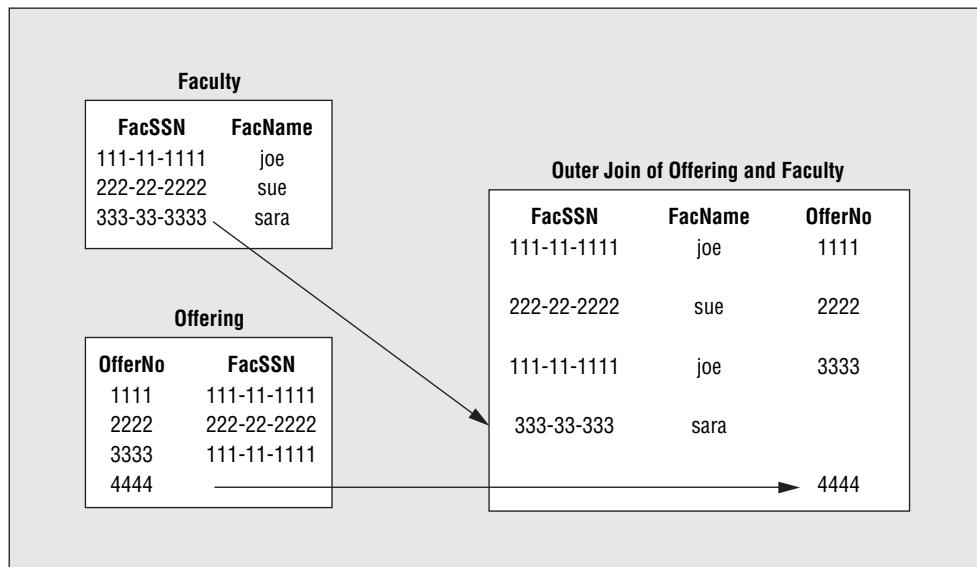
#### *Formulación visual de operaciones de enlace*

Con el fin de ofrecer ayuda para la formulación de consultas, muchos DBMS cuentan con un método visual para formular enlaces. Microsoft Access tiene una representación visual del operador de enlace en la ventana diseño de consulta. La figura 3.6 ilustra el enlace entre *Student* y *Enrollment* en *StdSSN* con el uso de la ventana diseño de consulta. Para formular este enlace basta con seleccionar las tablas. Access determina que la unión se hace en relación con la columna *StdSSN*. Access supone que la mayoría de los enlaces involucran la combinación de una clave primaria y una clave externa. En el caso de que Access seleccione la condición de enlace de manera incorrecta, usted puede escoger otra columna de enlace.

**FIGURA 3.6**  
Ventana de diseño de consultas que muestra un enlace entre Student y Enrollment



**FIGURA 3.7**  
Ejemplo de operación de enlace externo (*outer join*)



### 3.4.4 Operador de enlace externo (*outer join*)

El resultado de una operación de enlace incluye las filas que coinciden con la condición del enlace. En ocasiones es útil incluir tanto las filas que coinciden como los que no coinciden. Por ejemplo, quizás quiera conocer los cursos ofertados que tengan asignado un instructor, así como los que no lo tienen. En estas situaciones resulta útil el operador de enlace externo.

El operador del enlace externo ofrece la capacidad de preservar las filas que no coinciden en el resultado, así como incluir las filas que sí coinciden. La figura 3.7 ilustra un ejemplo de enlace externo entre las tablas *Faculty* y *Offering*. Observe que cada tabla tiene una fila que no coincide con ninguna fila de la otra tabla. La tercera fila de *Faculty* y la cuarta de *Offering* no tienen filas que coincidan con la otra tabla. En las filas que no coinciden se usan valores nulos para completar los valores de las columnas en la otra tabla. En la figura 3.7, los espacios en blanco (sin valores) representan valores nulos. La cuarta fila del resultado es la fila que no coincide con *Faculty* y tiene un valor nulo para la columna *OfferNo*. De forma análoga, la quinta fila del resultado contiene un valor nulo para las primeras dos columnas, ya que es una fila sin coincidencias de *Offering*.

**enlace externo completo (*full outer join*)**

un operador que genera las filas que coinciden (la parte del enlace) y las filas que no coinciden de ambas tablas.

**enlace externo de un solo lado (*one-sided outer join*)**

un operador que genera las filas que coinciden (la parte del enlace) y las filas que no coinciden de la tabla designada.

*Operadores de enlace completo versus operadores de enlace externo por un solo lado*

El operador de enlace externo (*outer join*) tiene dos variantes. El operador de enlace externo completo (*full outer join*) conserva las filas que no coincidan de las dos tablas iniciales. La figura 3.7 muestra un enlace externo completo, ya que las filas que no coinciden de ambas tablas se conservan en el resultado. Debido a que en ocasiones es útil conservar las filas que no coinciden de una de las tablas, se creó el operador de enlace externo de un solo lado (*one-sided outer join*). En la figura 3.7, sólo las cuatro primeras filas del resultado aparecerán para un enlace externo de un sólo lado que conserve las filas de la tabla *Faculty*. El último renglón no aparecerá, ya que es una fila que no coincide de la tabla *Offering*. De forma similar, para un enlace externo de un solo lado que conserve las filas de la tabla *Offering*, sólo aparecerán en el resultado las tres primeras filas y la última.

El enlace externo es útil en dos situaciones. Un enlace externo completo se puede usar para combinar dos tablas con renglones comunes y algunas columnas únicas. Por ejemplo, para combinar las tablas *Student* y *Faculty*, se puede usar un enlace externo completo para mostrar todas las columnas de todas las personas de la universidad. En la tabla 3.18, los primeros dos renglones son sólo del ejemplo de la tabla *Student* (tabla 3.16), mientras que las últimas dos filas son sólo del ejemplo de la tabla *Faculty* (tabla 3.17). Observe el uso de los valores nulos para las columnas de la otra tabla. La tercera fila de la tabla 3.18 es la fila común de las tablas de ejemplo *Faculty* y *Student*.

Un enlace externo de un solo lado puede ser útil cuando una tabla tiene valores nulos en la llave foránea. Por ejemplo, la tabla *Offering* (tabla 3.19) puede tener valores nulos en la columna *FacSSN* que representa los cursos que se ofrecen sin profesor asignado. Un enlace externo de un solo lado entre *Offering* y *Faculty* conserva las filas de *Offering* que no tienen asignado un *Faculty*, tal como se muestra en la tabla 3.20. Con un enlace natural no deberían aparecer la primera y la tercera filas de la tabla 3.20. Como verá en el capítulo 10, los enlaces de un solo lado pueden ser útiles en los formularios de captura.

**TABLA 3.16**  
Ejemplo de la tabla  
*Student*

StdSSN	StdLastName	StdMajor	StdClass
123-45-6789	WELLS	IS	FR
124-56-7890	NORBERT	FIN	JR
876-54-3210	COLAN	MS	SR

**TABLA 3.17**  
Ejemplo de la tabla  
*Faculty*

FacSSN	FacLastName	FacDept	FacRank
098-76-5432	VINCE	MS	ASST
543-21-0987	EMMANUEL	MS	PROF
876-54-3210	COLAN	MS	ASST

**TABLA 3.18** Resultado del enlace externo completo de las tablas *Student* y *Faculty*

StdSSN	StdLastName	StdMajor	StdClass	FacSSN	FacLastName	FacDept	FacRank
123-45-6789	WELLS	IS	FR				
124-56-7890	NORBERT	FIN	JR				
876-54-3210	COLAN	MS	SR	876-54-3210	COLAN	MS	ASST
				098-76-5432	VINCE	MS	ASST
				543-21-0987	EMMANUEL	MS	PROF

**TABLA 3.19**  
Ejemplo de la tabla  
*Offering*

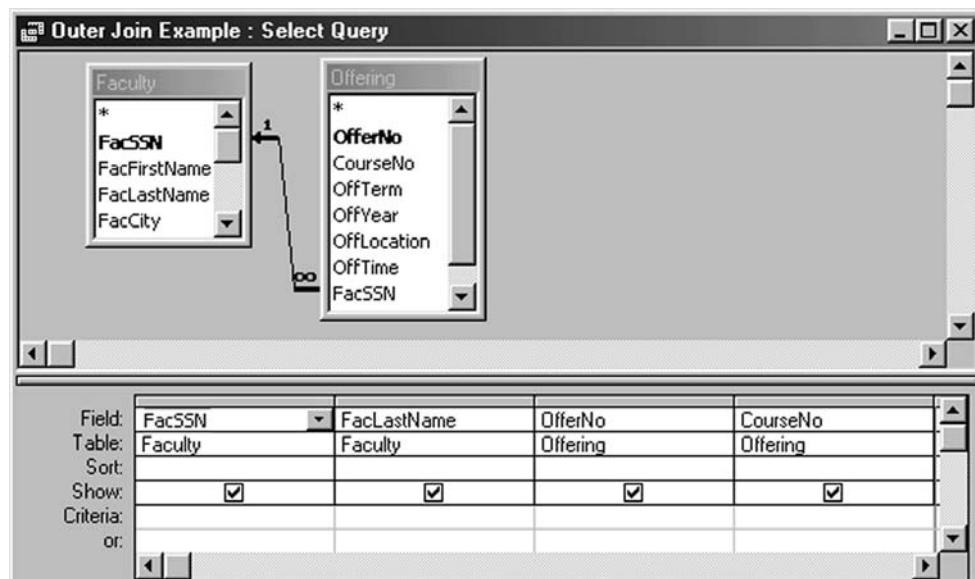
OfferNo	CourseNo	OffTerm	FacSSN
1111	IS320	SUMMER	
1234	IS320	FALL	098-76-5432
2222	IS460	SUMMER	
3333	IS320	SPRING	098-76-5432
4444	IS320	SPRING	543-21-0987

**TABLA 3.20** Resultado del enlace externo de un solo lado entre *Offering* (tabla 3.19) y *Faculty* (tabla 3.17)

OfferNo	CourseNo	OffTerm	Offering.FacSSN	Faculty.FacSSN	FacLastName	FacDept	FacRank
1111	IS320	SUMMER					
1234	IS320	FALL	098-76-5432	098-76-5432	VINCE	MS	ASST
2222	IS460	SUMMER					
3333	IS320	SPRING	098-76-5432	098-76-5432	VINCE	MS	ASST
4444	IS320	SPRING	543-21-0987	543-21-0987	EMMANUEL	MS	PROF

**FIGURA 3.8**

Ventana de diseño de consulta que muestra un enlace externo de un solo lado que conserva la tabla *Offering*



#### Formulación visual de las operaciones de enlace externo

Como ayuda para la formulación de consultas, muchos DBMS proporcionan un modo visual para formular enlaces externos. Microsoft Access proporciona una representación visual del operador de enlace de un solo lado en la ventana de diseño de consulta. La figura 3.8 ilustra un enlace externo de un solo lado que conserva las filas de la tabla *Offering*. La flecha que va de la tabla *Offering* a la tabla *Faculty* significa que las filas que no coincidan de *Offering* se conservan en el resultado. Cuando se combinan las tablas *Faculty* y *Offering*, Microsoft Access proporciona tres alternativas: (1) mostrar sólo las filas que coinciden (un enlace); (2) mostrar las filas que coinciden y las que no coinciden de *Faculty*; y (3) mostrar las filas que coinciden y las que no coinciden de *Offering*. La opción (3) se muestra en la figura 3.8. La opción (1) es similar a la figura 3.6. La opción (2) tendría la flecha desde *Faculty* hasta *Offering*.

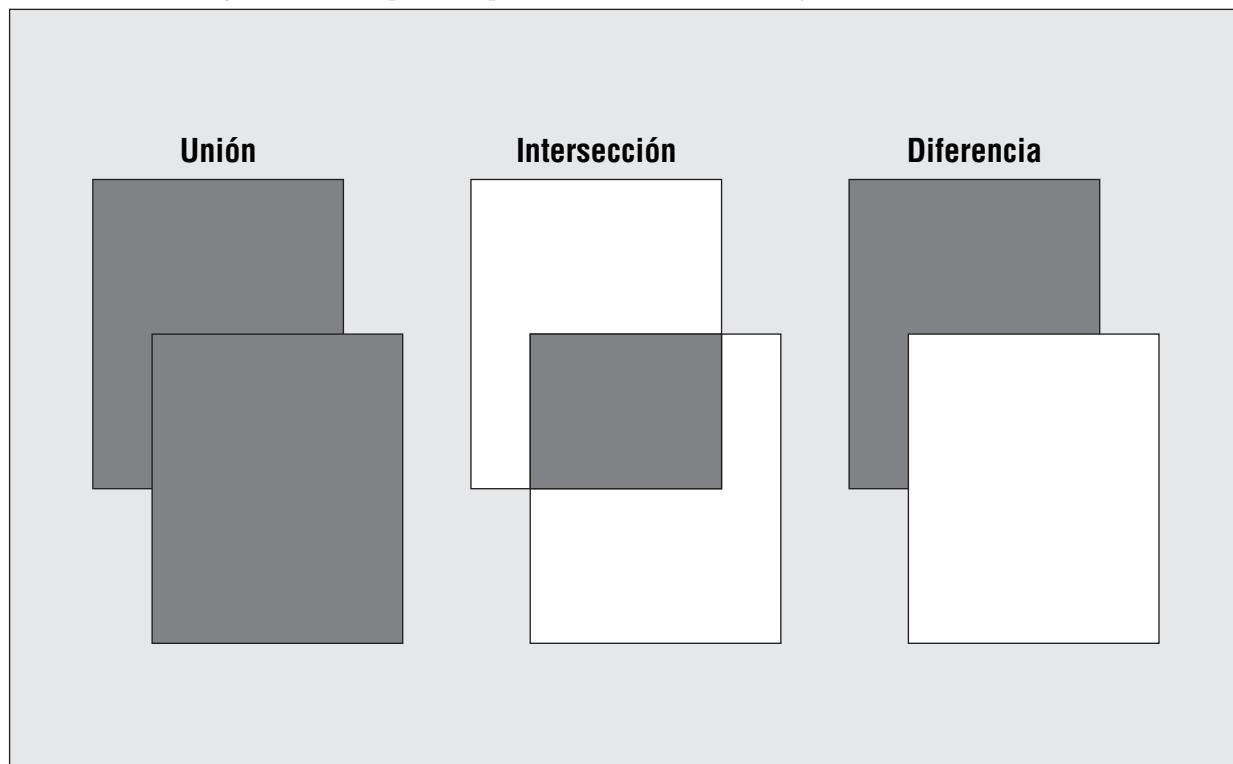
#### operadores tradicionales de conjuntos

el operador unión genera una tabla que contiene las filas de cualquiera de las tablas de entrada. El operador intersección genera una tabla que contiene las filas que son comunes entre las tablas iniciales. El operador de diferencia genera una tabla que contiene las filas de la primera tabla, pero que no están en la segunda.

#### 3.4.5 Operadores de unión, intersección y diferencia

Los operadores de tablas unión, intersección y diferencia son semejantes a los operadores de conjuntos tradicionales. Los operadores tradicionales de conjuntos se usan para determinar todos los miembros de dos conjuntos (unión), miembros comunes de dos conjuntos (intersección) y miembros que sólo están en un conjunto (diferencia), tal como se ilustra en la figura 3.9.

Los operadores de unión, intersección y diferencia se aplican a las filas de una tabla pero operan de la misma forma que los operadores de conjuntos tradicionales. Una operación de unión extrae todas las filas de cualquier tabla. Por ejemplo, un operador de unión que se aplica a dos tablas de estudiantes de distintas universidades puede encontrar todas las filas de estudiantes. Una operación de intersección extrae sólo las filas comunes. Por ejemplo, una operación de intersección puede determinar los estudiantes que asisten a ambas universidades. Una operación de diferencia extrae las filas de la primera tabla que no están en la segunda tabla. Por ejemplo, una operación de diferencia puede determinar qué estudiantes asisten sólo a una universidad.

**FIGURA 3.9** Diagramas de Venn para los operadores tradicionales de conjuntos**TABLA 3.21**  
Tabla *Student1*

StdSSN	StdLastName	StdCity	StdState	StdMajor	StdClass	StdGPA
123-45-6789	WELLS	SEATTLE	WA	IS	FR	3.00
124-56-7890	NORBERT	BOTHELL	WA	FIN	JR	2.70
234-56-7890	KENDALL	TACOMA	WA	ACCT	JR	3.50

**TABLA 3.22**  
Tabla *Student2*

StdSSN	StdLastName	StdCity	StdState	StdMajor	StdClass	StdGPA
123-45-6789	WELLS	SEATTLE	WA	IS	FR	3.00
995-56-3490	BAGGINS	AUSTIN	TX	FIN	JR	2.90
111-56-4490	WILLIAMS	SEATTLE	WA	ACCT	JR	3.40

### compatibilidad de unión

un requerimiento de las tablas iniciales para los operadores tradicionales de conjuntos. Cada tabla debe tener el mismo número de columnas y cada columna correspondiente debe tener un tipo de dato compatible.

### Compatibilidad de unión

La compatibilidad es un concepto nuevo para los operadores de tablas en comparación con los operadores tradicionales de conjuntos. Con los operadores de tablas, ambas tablas deben ser compatibles, ya que se comparan todas las columnas. La compatibilidad de unión significa que cada tabla debe tener el mismo número de columnas y cada columna correspondiente debe tener un tipo de dato compatible. La compatibilidad de unión puede ser confusa, ya que involucra la correspondencia posicional de las columnas. Esto es, las primeras columnas de las dos tablas deben tener tipos de datos compatibles, las segundas columnas deben tener tipos de datos compatibles, etcétera.

Para ilustrar los operadores de unión, intersección y diferencia, aplíquemoslos a las tablas *Student1* y *Student2* (tablas 3.21 y 3.22). Estas tablas son compatibles con el operador de unión porque tienen columnas idénticas en el mismo orden. Los resultados de los operadores de unión, intersección y diferencia se muestran en las tablas 3.23 a 3.25, respectivamente. Aunque podemos identificar que las dos filas son idénticas observando solamente *StdSSN*, todas las columnas se comparan debido a la forma en que se asignan los operadores.

**TABLA 3.23**  
**Student1 UNIÓN**  
**Student2**

StdSSN	StdLastName	StdCity	StdState	StdMajor	StdClass	StdGPA
123-45-6789	WELLS	SEATTLE	WA	IS	FR	3.00
124-56-7890	NORBERT	BOTHELL	WA	FIN	JR	2.70
234-56-7890	KENDALL	TACOMA	WA	ACCT	JR	3.50
995-56-3490	BAGGINS	AUSTIN	TX	FIN	JR	2.90
111-56-4490	WILLIAMS	SEATTLE	WA	ACCT	JR	3.40

**TABLA 3.24**  
**Student1**  
**INTERSECCIÓN**  
**Student2**

StdSSN	StdLastName	StdCity	StdState	StdMajor	StdClass	StdGPA
123-45-6789	WELLS	SEATTLE	WA	IS	FR	3.00

**TABLA 3.25**  
**Student1**  
**DIFERENCIA**  
**Student2**

StdSSN	StdLastName	StdCity	StdState	StdMajor	StdClass	StdGPA
124-56-7890	NORBERT	BOTHELL	WA	FIN	JR	2.70
234-56-7890	KENDALL	TACOMA	WA	ACCT	JR	3.50

Observe que el resultado de *Student1 DIFERENCIA Student2* no sería el mismo que el de *Student2 DIFERENCIA Student1*. El resultado de este último (*Student2 DIFERENCIA Student1*) corresponde al segundo y tercer renglones de *Student2* (renglones que están en *Student2* pero no en *Student1*).

Debido al requerimiento de compatibilidad del operador de unión, los operadores de unión, intersección y diferencia no se usan tan a menudo como otros operadores. Sin embargo, estos operadores tienen algunos usos especializados e importantes. Uno de ellos es combinar tablas distribuidas en múltiples ubicaciones. Por ejemplo, suponga que existe una tabla de estudiantes de la Big State University (*BSUStudent*) y una tabla de estudiantes de University of Big State (*UBSStudent*). Debido a que estas tablas tienen columnas idénticas, se pueden aplicar los operadores de conjuntos tradicionales. Para encontrar estudiantes que van a cualquiera de las dos universidades, debe utilizar *UBSStudent UNIÓN BSUStudent*. Para encontrar estudiantes que sólo vayan a Big State, usted debe utilizar *BSUStudent DIFERENCIA UBSStudent*. Para encontrar estudiantes que asistan a las dos universidades, debe utilizar *UBSStudent INTERSECCIÓN BSUStudent*. Observe que la tabla resultante de cada operación tiene el mismo número de columnas que las dos tablas iniciales.

Los operadores tradicionales también son útiles si hay tablas que sean similares, pero no compatibles con el operador de unión. Por ejemplo, las tablas *Student* y *Faculty* tienen algunas columnas compatibles (*StdSSN* con *FacSSN* *StdLastName* con *FacLastName* y *StdCity* con *FacCity*), pero otras distintas. Los operadores de compatibilidad de unión se pueden usar si las tablas *Student* y *Faculty* se hacen primero compatibles utilizando el operador de proyección presentado en la sección 3.4.1.

### 3.4.6 Operador resumir (*summarize*)

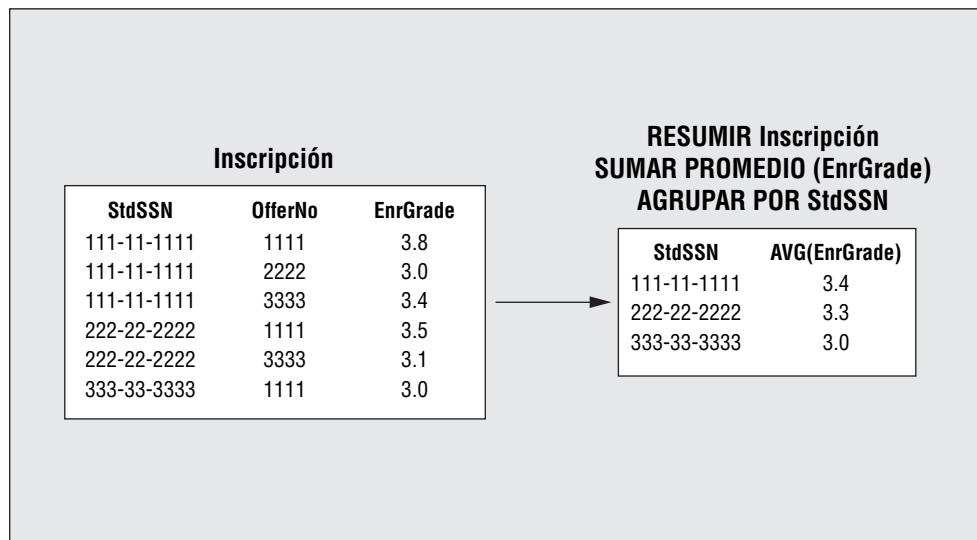
El operador resumir es un poderoso factor para la toma de decisiones. Debido a que las tablas pueden incluir muchas filas, generalmente es útil observar las estadísticas de los grupos de filas en lugar de las filas de manera individual. El operador *resumir* permite que grupos de filas se compriman o resuman para un valor calculado. Casi cualquier tipo de función estadística se puede usar para resumir grupos de filas. Como éste no es un libro de estadística, usaremos funciones estadísticas sencillas, como contar, valor mínimo, valor máximo, promedio y suma.

El operador resumir comprime una tabla al reemplazar grupos de filas por filas individuales que contienen valores calculados. Una función estadística o agregada se usa para los valores calculados. La figura 3.10 ilustra una operación resumida para un ejemplo de tabla de inscripciones. La tabla inicial se agrupa bajo la columna *StdSSN*. Cada grupo de filas se reemplaza por el promedio de la columna del grado.

#### resumir

un operador que genera una tabla con filas que resumen las filas de la tabla inicial. Las funciones agregadas se usan para resumir las filas de la tabla inicial.

**FIGURA 3.10**  
Ejemplo de la operación resumir



**TABLA 3.26** Ejemplo de la tabla *Faculty*

FacSSN	FacLastName	FacDept	FacRank	FacSalary	FacSupervisor	FacHireDate
098-76-5432	VINCE	MS	ASST	\$35,000	654-32-1098	01-Apr-95
543-21-0987	EMMANUEL	MS	PROF	\$120,000		01-Apr-96
654-32-1098	FIBON	MS	ASSC	\$70,000	543-21-0987	01-Apr-94
765-43-2109	MACON	FIN	PROF	\$65,000		01-Apr-97
876-54-3210	COLAN	MS	ASST	\$40,000	654-32-1098	01-Apr-99
987-65-4321	MILLS	FIN	ASSC	\$75,000	765-43-2109	01-Apr-00

**TABLA 3.27**  
Tabla resultante de  
RESUMIR *Faculty*  
SUMAR PROMEO-  
DIO (*FacSalary*)  
AGRUPAR POR  
*FacDept*

FacDept	FacSalary
MS	\$66,250
FIN	\$70,000

Otro ejemplo: la tabla 3.27 muestra el resultado de la operación resumir en la tabla de ejemplo *Faculty* de la tabla 3.26. Observe que el resultado contiene una fila por cada valor de la columna agrupadora, *FacDept*.

El operador resumir puede incluir valores adicionales calculados (que también muestran el salario mínimo, por ejemplo) y columnas agrupadoras adicionales (también agrupadas en *FacRank*, por ejemplo). Cuando se agrupan varias columnas, cada resultado muestra una combinación de valores para las columnas agrupadoras.

**dividir (divide)**  
un operador que genera una tabla en la cual los valores de una columna de una de las tablas de entrada se asocian con todos los valores de una columna de una segunda tabla de entrada.

### 3.4.7 Operador dividir (divide)

El operador dividir es un operador más especializado y difícil de usar que un enlace, ya que el requerimiento de valores que coincidan en la división es más estricto que en el enlace. Por ejemplo, un operador de enlace se usa para extraer los cursos tomados por *cualquier* estudiante. Un operador de división se requiere para extraer los cursos tomados por *todos* (o cada uno) de los estudiantes. Debido a que el operador dividir tiene condiciones de coincidencia más estrictas, su uso no es tan amplio como el del operador de enlace, al tiempo que es más difícil de entender. Cuando es apropiado, el operador dividir proporciona una poderosa forma de combinar tablas.

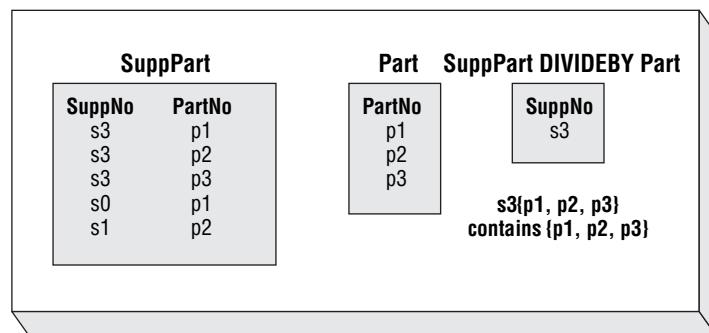
El operador dividir de las tablas es de cierta forma análogo al operador dividir de los números. En una división numérica, el objetivo es encontrar las veces que un número se encuentra dentro de otro. En la división de tablas, el objetivo es encontrar valores de una columna que contenga *todos* los valores de otra columna. Dicho de otra forma, el operador dividir encuentra los valores de una columna que están asociados con *todos* los valores de otra columna.

Para comprender de manera más concreta la forma en que trabaja el operador dividir, considere un ejemplo con las tablas de muestra *Part* y *SuppPart* (parte-proveedor, *supplier-part*) como se ilustra en la figura 3.11. El operador dividir usa dos tablas iniciales. La primera tabla (*SuppPart*) tiene dos columnas (una tabla binaria) y la segunda tabla (*Part*) tiene una columna (una tabla unitaria).<sup>10</sup> La tabla resultante tiene una columna en la cual los valores provienen de la primera columna de la tabla binaria. La tabla resultante de la figura 3.11 muestra a los proveedores que ofrecen cada una de las partes. El valor s3 aparece en la salida, ya que está asociado con *cada uno* de los valores de la tabla *Part*. Dicho de otra forma, el conjunto de valores asociado con s3 contiene el conjunto de valores de la tabla *Part*.

Para comprender el operador dividir de otra forma, reescriba la tabla *SuppPart* como tres filas que usan los símbolos < > para enmarcar una fila: <s3, {p1, p2, p3}>, <s0, {p1}>, <s1, {p2}>. Reescriba la tabla *Part* como un conjunto: {p1, p2, p3}. El valor s3 está en la tabla de resultado, ya que su conjunto de valores de la segunda columna {p1, p2, p3} contiene los valores de la segunda tabla {p1, p2, p3}. Los otros valores de *SuppNo* (s0 y s1) no están en el resultado porque no están asociados con todos los valores de la tabla *Part*.

En un ejemplo utilizando las tablas de la base de datos de la universidad, la tabla 3.30 muestra el resultado de una operación de división que involucra las tablas de ejemplo *Enrollment* (tabla 3.28) y *Student* (tabla 3.29). El resultado muestra los cursos en los que se encuentra inscrito cada estudiante. Sólo *OfferNo* 4235 tiene a sus tres estudiantes inscritos.

**FIGURA 3.11**  
Ejemplo de la operación dividir



**TABLA 3.28**  
Ejemplo de la tabla *Enrollment*

OfferNo	StdSSN
1234	123-45-6789
1234	234-56-7890
4235	123-45-6789
4235	234-56-7890
4235	124-56-7890
6321	124-56-7890

**TABLA 3.29**  
Ejemplo de la tabla *Student*

StdSSN
123-45-6789
124-56-7890
234-56-7890

**TABLA 3.30**  
Resultado de *Enrollment* DIVIDIDO ENTRE *Student*

OfferNo
4235

<sup>10</sup> La operación dividir entre puede generalizarse para que trabaje con tablas de entrada que contengan más columnas. Sin embargo, los detalles no son importantes en este libro.

**TABLA 3.31**

**Resumen del significado de los operadores de álgebra relacional**

Operador	Significado
Restricción (selección, <i>select</i> )	Extrae las filas que satisfacen una condición específica.
Proyección ( <i>project</i> )	Extrae determinadas columnas.
Producto ( <i>product</i> )	Construye una tabla a partir de dos tablas que contiene todas las combinaciones posibles de filas, una por cada una de las dos tablas.
Unión ( <i>union</i> )	Construye una tabla formada por todas las filas que aparecen en cualquiera de las dos tablas.
Intersección ( <i>intersect</i> )	Construye una tabla formada por todas las filas que aparecen en ambas tablas.
Diferencia ( <i>difference</i> )	Construye una tabla formada por todas las filas que aparecen en la primera tabla pero no en la segunda.
Enlace ( <i>join</i> )	Extrae las filas del producto de dos tablas, de tal forma que dos filas de entrada contribuyan a cualquier fila de salida que satisfaga una determinada condición.
Enlace externo ( <i>outer join</i> )	Extrae las filas que coinciden [la parte del enlace ( <i>join</i> )] en dos tablas y las filas que no coinciden de una o de las dos tablas.
División ( <i>divide</i> )	Construye una tabla formada por todos los valores de columna de una tabla binaria (dos columnas) que coinciden (en la otra columna) con todos los valores de una tabla unitaria (una columna).
Resumen ( <i>summarize</i> )	Organiza una tabla según las columnas que se especifican como agrupación. Los cálculos agregados se hacen sobre cada uno de los valores de las columnas agrupadas.

**TABLA 3.32**

**Resumen del uso de operadores de álgebra relacional**

Operador	Notas
Unión ( <i>union</i> )	Las tablas de entrada deben ser compatibles con el operador unión.
Diferencia ( <i>difference</i> )	Las tablas de entrada deben ser compatibles con el operador unión.
Intersección ( <i>intersection</i> )	Las tablas de entrada deben ser compatibles con el operador unión.
Producto ( <i>product</i> )	Conceptualmente, implica el operador de enlace.
Restricción (selección, <i>restrict-select</i> )	Usa una expresión lógica.
Proyección ( <i>project</i> )	Si es necesario elimina las filas duplicadas.
Enlace ( <i>join</i> )	Sólo las filas que coinciden son parte del resultado. El enlace natural elimina el enlace de una columna.
Enlace externo ( <i>outer join</i> )	Conserva en el resultado tanto las filas que coinciden como las que no coinciden. Utiliza los valores nulos para algunas columnas de las filas que no coinciden.
División ( <i>divide</i> )	Un operador más fuerte que el enlace, pero que se usa con menor frecuencia.
Resumen ( <i>summarize</i> )	Especifica las columnas de agrupación, si las hay, y las funciones agregadas.

### 3.4.8 Resumen de operadores

Para ayudarle a recordar los operadores de álgebra relacional, las tablas 3.31 y 3.32 ofrecen un resumen adecuado del significado y uso de cada operador. Si lo desea, puede hacer referencia a estas tablas cuando estudie la formulación de consultas en los siguientes capítulos.

## Reflexión final

El capítulo 3 introdujo el modelo de datos relacional como un preludio al desarrollo de consultas, formularios y reportes con bases de datos relacionales. Como primer paso para trabajar con bases de datos relacionales, debe comprender la terminología básica y las reglas de integridad. También debe ser capaz de leer las definiciones de tablas en SQL y en otros formatos propietarios. Para hacer una consulta efectiva a una base de datos relacional debe comprender las conexiones entre tablas. La mayoría de las consultas involucran varias tablas que usan relaciones definidas por la integridad referencial. Una representación gráfica como la ventana de relaciones de Microsoft

Access proporciona una poderosa herramienta para conceptualizar la restricción de integridad referencial. Cuando se desarrollan aplicaciones que pueden modificar una base de datos es importante respetar las reglas de acción de las filas referenciadas.

La parte final de este capítulo describe los operadores de álgebra relacional. En este punto, debe comprender el propósito de cada operador, el número de tablas de entrada y otras entradas que se utilicen. No necesita escribir fórmulas complicadas que combinen operadores. Con el tiempo debe sentirse tranquilo al comprender sentencias tales como: “Escriba una sentencia SQL SELECT para enlazar tres tablas”. La sentencia SELECT se describirá en los capítulos 4 y 9, pero es importante aprender la idea básica de un enlace. Cuando aprenda a extraer datos utilizando la sentencia SQL SELECT en el capítulo 4, es posible que quiera revisar nuevamente este capítulo. Para ayudarle a recordar los puntos más importantes de los operadores, la última sección de este capítulo presenta varios resúmenes.

La comprensión de los operadores mejorará su conocimiento de SQL y sus habilidades en la formulación de consultas. El significado de las consultas de SQL se puede entender como si fuesen operaciones de álgebra relacional. El capítulo 4 proporciona un diagrama de flujo que muestra esta correspondencia. Por esta razón, el álgebra relacional proporciona un criterio para medir los lenguajes comerciales: los lenguajes comerciales deben proporcionar al menos la misma habilidad de extracción que los operadores de álgebra relacional.

## Revisión de conceptos

- Tablas: encabezado y cuerpo.
- Llaves primarias y regla de integridad de la entidad.
- Llaves foráneas, regla de integridad referencial y valores que coinciden.
- Visualización de las restricciones de la integridad referencial.
- Representación del modelo relacional de las relaciones 1-M, relaciones M-N y relaciones autorreferenciales.
- Acciones sobre filas referenciadas: cascada, nulificar, restricción, por omisión.
- Subconjunto de operadores: restricción (*select*) y proyección.
- Operador de enlace para combinar dos tablas usando una condición para comparar las columnas de enlace.
- Enlace natural usando la igualdad para el operador, enlaza columnas con el mismo nombre no calificado y elimina una de las columnas del enlace.
- Operador más usado para combinar tablas: enlace natural.
- Operadores menos usados para combinar tablas: enlace externo completo, enlace externo de un solo lado, división.
- El operador de enlace externo extiende al operador de enlace conservando las filas sin coincidencia.
- El operador de enlace externo de un solo lado conserva las filas que no coinciden de una de las tablas de entrada.
- El enlace externo completo conserva las filas que no coinciden de ambas tablas.
- Operadores tradicionales de conjuntos: unión, intersección, diferencia, producto cruzado extendido.
- Compatibilidad del operador unión para comparar filas para la unión, intersección y diferencia.
- Operador complejo: operador dividir para encontrar coincidencias de un subconjunto de filas.
- El operador resumir reemplaza grupos de filas con filas resumidas.

## Preguntas

1. ¿Por qué la creación de una tabla es similar a escribir el capítulo de un libro?
2. ¿Con qué terminología de las bases de datos relacionales se siente usted más cómodo? ¿Por qué?
3. ¿Cuál es la diferencia entre una llave primaria y una llave candidata?

4. ¿Cuál es la diferencia entre una llave candidata y una superllave?
5. ¿Qué es un valor nulo?
6. ¿Cuál es el motivo para la regla de integridad de la entidad?
7. ¿Cuál es el motivo para la regla de integridad referencial?
8. ¿Cuál es la relación entre la regla de integridad referencial y las llaves foráneas?
9. ¿Cómo se indican las llaves candidatas que no son llaves primarias en la sentencia CREATE TABLE?
10. ¿Cuál es la ventaja de utilizar nombres para las restricciones cuando se definen las restricciones de una llave primaria, llave candidata o restricciones de integridad referencial en las sentencias CREATE TABLE?
11. ¿Cuándo no está permitido almacenar valores nulos en las llaves foráneas?
12. ¿Cuál es el propósito de un diagrama de base de datos, como la ventana relaciones de Access?
13. ¿Cómo se representa una relación 1-M en el modelo relacional?
14. ¿Cómo se representa una relación M-N en el modelo relacional?
15. ¿Qué es una relación autorreferencial?
16. ¿Cómo se representa una relación autorreferencial en el modelo relacional?
17. ¿Qué es una fila referenciada?
18. ¿Cuáles son las dos acciones de las filas referenciadas que pueden afectar a las filas relacionadas de una tabla hija?
19. ¿Cuáles son las posibles acciones sobre las filas relacionadas después de haber borrado una fila referenciada o haber actualizado su llave primaria?
20. ¿Por qué la acción de restricción de filas referenciadas es más común que la acción en cascada?
21. ¿En qué ocasión no se permite la acción nulificar?
22. ¿Por qué se estudian los operadores del álgebra relacional?
23. ¿Por qué se usan tanto los operadores de restricción y proyección?
24. Explique por qué los operadores de unión, intersección y diferencia de las tablas difieren de los operadores tradicionales de conjuntos.
25. ¿Por qué es tan importante el operador de enlace para extraer información útil?
26. ¿Cuál es la relación entre el enlace y los operadores extendidos de producto cruz?
27. ¿Por qué se usa tan esporádicamente el operador extendido de producto cruz?
28. ¿Qué sucede con las filas que no coinciden con el operador de enlace?
29. ¿Qué sucede con las filas que no coinciden con el operador de enlace externo?
30. ¿Cuál es la diferencia entre el enlace externo completo y el enlace externo de un solo lado?
31. Defina una situación de toma de decisiones que pudiera requerir del operador resumir.
32. ¿Qué es una función agregada?
33. ¿Cómo se usan las columnas agrupadas con el operador resumir?
34. ¿Por qué no se usa el operador división tanto como el de enlace?
35. ¿Cuáles son los requerimientos de compatibilidad de unión?
36. ¿Cuáles son los requerimientos del operador de unión natural?
37. ¿Por qué es tan usado el operador de enlace natural para combinar tablas?
38. ¿Cómo es que herramientas visuales como la de diseño de consultas de Microsoft Access facilitan la formulación de operaciones de enlace?

## Problemas

Los problemas usan las tablas *Customer*, *OrderTbl* y *Employee* de la base de datos simplificada de captura de órdenes. Los capítulos 4 y 10 extienden la base de datos para ampliar su utilidad. La tabla *Customer* registra a los clientes que han hecho órdenes. La tabla *OrderTbl* contiene hechos básicos sobre las órdenes de los clientes. La tabla *Employee* contiene los hechos de los empleados que reciben las órdenes. Las llaves primarias de las tablas son *CustNo* para *Customer*, *EmpNo* para *Employee* y *OrdNo* para *OrderTbl*.

**Customer**

CustNo	CustFirstName	CustLastName	CustCity	CustState	CustZip	CustBal
C0954327	Sheri	Gordon	Littleton	CO	80129-5543	\$230.00
C1010398	Jim	Glussman	Denver	CO	80111-0033	\$200.00
C2388597	Beth	Taylor	Seattle	WA	98103-1121	\$500.00
C3340959	Betty	Wise	Seattle	WA	98178-3311	\$200.00
C3499503	Bob	Mann	Monroe	WA	98013-1095	\$0.00
C8543321	Ron	Thompson	Renton	WA	98666-1289	\$85.00

**Employee**

EmpNo	EmpFirstName	EmpLastName	EmpPhone	EmpEmail
E1329594	Landi	Santos	(303) 789-1234	LSantos@bigco.com
E8544399	Joe	Jenkins	(303) 221-9875	JJenkins@bigco.com
E8843211	Amy	Tang	(303) 556-4321	ATang@bigco.com
E9345771	Colin	White	(303) 221-4453	CWhite@bigco.com
E9884325	Thomas	Johnson	(303) 556-9987	TJohnson@bigco.com
E9954302	Mary	Hill	(303) 556-9871	MHill@bigco.com

**OrderTbl**

OrdNo	OrdDate	CustNo	EmpNo
O1116324	01/23/2007	C0954327	E8544399
O2334661	01/14/2007	C0954327	E1329594
O3331222	01/13/2007	C1010398	
O2233457	01/12/2007	C2388597	E9884325
O4714645	01/11/2007	C2388597	E1329594
O5511365	01/22/2007	C3340959	E9884325
O7989497	01/16/2007	C3499503	E9345771
O1656777	02/11/2007	C8543321	
O7959898	02/19/2007	C8543321	E8544399

1. Escriba una sentencia CREATE TABLE para la tabla *Customer*. Elija los tipos de datos apropiados para el DBMS usado en su curso. Observe que la columna *CustBal* contiene datos numéricos. Los símbolos de monedas no se almacenan en la base de datos. Se requieren las columnas *CustFirstName* y *CustLastName* (no nulas).
2. Escriba una sentencia CREATE TABLE para la tabla *Employee*. Elija los tipos de datos apropiados para el DBMS usado en su curso. Se requieren las columnas *EmpFirstName*, *EmpLastName* y *EmpEMail* (no nulas).
3. Escriba una sentencia CREATE TABLE para la tabla *OrderTbl*. Elija los tipos de datos apropiados para el DBMS usado en su curso. La columna *OrdDate* es requerida (no nula).
4. Identifique las llaves foráneas y dibuje un diagrama de relaciones para la base de datos simplificada de captura de órdenes. La columna *CustNo* hace referencia a la tabla *Customer* y la columna *EmpNo* hace referencia a la tabla *Employee*. Para cada relación, identifique la tabla madre y la tabla hija.
5. Extienda su sentencia CREATE TABLE del problema 3, incluyendo las restricciones de integridad referencial. Está restringida la actualización y la eliminación de las filas relacionadas.
6. A partir de la revisión de los datos del ejemplo y de su sentido común para la captura de órdenes comerciales, ¿se permiten los valores nulos para las llaves foráneas de la tabla *OrderTbl*? ¿Por qué sí o por qué no? Extienda la sentencia CREATE TABLE del problema 5 para hacer cumplir las restricciones de los valores nulos, en caso de que existan.

7. Extienda la sentencia CREATE TABLE de la tabla *Employee* (problema 2) con una restricción única para *EmpEMail*. Utilice una cláusula de restricción nombrada para la restricción única.
8. Muestre el resultado de la operación de restricción (*restrict*) que enliste las órdenes de febrero de 2007.
9. Muestre el resultado de la operación de restricción que enliste a los clientes que viven en Seattle, WA.
10. Muestre el resultado de la operación de proyección (*project*) que enliste las columnas *CustNo*, *CustFirstName* y *CustLastName* de la tabla *Customer*.
11. Muestre el resultado de la operación de proyección que enliste las columnas *CustCity* y *CustState* de la tabla *Customer*.
12. Muestre el resultado de un enlace natural (*natural join*) que combine las tablas *Customer* y *OrderTbl*.
13. Muestre los pasos para deducir el enlace natural del problema 10. ¿Cuántas filas y columnas se encuentran en el paso del producto cruzado extendido?
14. Muestre el resultado de un enlace natural de las tablas *Employee* y *OrderTbl*.
15. Muestre el resultado de un enlace externo de un solo lado entre las tablas *Employee* y *OrderTbl*. Consérve las filas de la tabla *OrderTbl* en el resultado.
16. Muestre el resultado de un enlace externo completo entre las tablas *Employee* y *OrderTbl*.
17. Muestre el resultado de una operación de restricción hecha sobre *Customer* en donde la condición *CustCity* sea igual a “Denver” o “Seattle”, seguida de la operación de proyección para conservar las columnas *CustNo*, *CustFirstName*, *CustLastName* y *CustCity*.
18. Muestre el resultado del enlace natural que combine las tablas *Customer* y *OrderTbl*, seguida de una operación de restricción para conservar sólo a los clientes de Colorado (*CustState* = “CO”).
19. Muestre el resultado de una operación de resumen (*summarize*) en la tabla *Customer*. La columna de agrupación es *CustState* y el cálculo agregado es COUNT. COUNT muestra el número de columnas con el mismo valor en la columna de agrupación.
20. Muestre el resultado de una operación de resumen en la tabla *Customer*. La columna de agrupación es *CustState* y el cálculo agregado son los valores mínimo y máximo de *CustBal*.
21. ¿Qué tablas se requieren para mostrar las columnas *CustLastName*, *EmpLastName* y *OrdNo* en la tabla resultante?
22. Extienda el diagrama de relaciones del problema 4 agregando dos tablas (*OrdLine* y *Product*). Las sentencias parciales CREATE TABLE para las llaves primarias y las restricciones de integridad referencial se muestran a continuación:

```

CREATE TABLE Product . . . PRIMARY KEY (ProdNo)
CREATE TABLE OrdLine . . . PRIMARY KEY (OrdNo, ProdNo)
    FOREIGN KEY (OrdNo) REFERENCES Order
    FOREIGN KEY (ProdNo) REFERENCES Product
  
```

23. Extienda el diagrama de relaciones del problema 22 agregando una llave foránea en la tabla *Employee*. La llave foránea *SupEmpNo* es el número de empleado del supervisor. Por lo tanto, la columna *SupEmpNo* hace referencia a la tabla *Employee*.
24. ¿Qué operador de álgebra relacional se usa para encontrar los productos que se encuentran en *cada* orden? ¿Qué operador de álgebra relacional se usa para encontrar los productos que se encuentran en *cualquier* orden?
25. ¿Las tablas *Customer* y *Employee* son compatibles con respecto al operador de unión? ¿Por qué sí o por qué no?
26. Utilizando la base de datos del problema 23, ¿qué tablas se deben combinar para listar los nombres de los productos de la orden número O1116324?
27. Utilizando la base de datos del problema 23, ¿qué tablas se deben combinar para listar los nombres de los productos ordenados por el cliente C0954327?
28. Utilizando la base de datos del problema 23, ¿qué tablas se deben combinar para listar los nombres de los productos ordenados por el cliente Sheri Gordon?
29. Utilizando la base de datos del problema 23, ¿qué tablas se deben combinar para listar los números de órdenes levantadas por los clientes que viven en Colorado?
30. Utilizando la base de datos del problema 23, ¿qué tablas se deben combinar para listar los nombres de los productos que aparecen en una orden levantada por el empleado Landi Santos?

## Referencias para ampliar su estudio

Codd definió el modelo relacional en un documento de gran importancia en 1970. Su documento inspiró los proyectos de investigación de los laboratorios de IBM y de la Universidad de California en Berkeley, lo que condujo a los DBMS comerciales. Date (2003) proporciona una sintaxis para el álgebra relacional. Elmasri y Navathe (2004) proporcionan un tratamiento más teórico del modelo relacional, en especial de álgebra relacional.

### Apéndice 3.A

## Sentencias CREATE TABLE para la base de datos de la universidad

Las siguientes son las sentencias CREATE TABLE para las tablas de la base de datos de la universidad (tablas 3.1, 3.3, 3.4, 3.6 y 3.7). Los nombres estándar de los tipos de datos pueden variar según el DBMS. Por ejemplo, Microsoft Access SQL soporta el tipo de dato TEXT en lugar de CHAR y VARCHAR. En Oracle, debe usar VARCHAR2 en lugar de VARCHAR.

```

CREATE TABLE Student
( StdSSN           CHAR(11),
  StdFirstName    VARCHAR(50)  CONSTRAINT StdFirstNameRequired NOT NULL,
  StdLastName     VARCHAR(50)  CONSTRAINT StdLastNameRequired NOT NULL,
  StdCity          VARCHAR(50)  CONSTRAINT StdCityRequired NOT NULL,
  StdState         CHAR(2)      CONSTRAINT StdStateRequired NOT NULL,
  StdZip           CHAR(10)     CONSTRAINT StdZipRequired NOT NULL,
  StdMajor          CHAR(6),
  StdClass          CHAR(2),
  StdGPA            DECIMAL(3,2),
  CONSTRAINT PKStudent PRIMARY KEY (StdSSN) )

```

```

CREATE TABLE Course
( CourseNo         CHAR(6),
  CrsDesc          VARCHAR(250) CONSTRAINT CrsDescRequired NOT NULL,
  CrsUnits         INTEGER,
  CONSTRAINT PKCourse PRIMARY KEY (CourseNo),
  CONSTRAINT UniqueCrsDesc UNIQUE (CrsDesc) )

```

```

CREATE TABLE Faculty
( FacSSN           CHAR(11),
  FacFirstName    VARCHAR(50)  CONSTRAINT FacFirstNameRequired NOT NULL,
  FacLastName     VARCHAR(50)  CONSTRAINT FacLastNameRequired NOT NULL,
  FacCity          VARCHAR(50)  CONSTRAINT FacCityRequired NOT NULL,
  FacState         CHAR(2)      CONSTRAINT FacStateRequired NOT NULL,
  FacZipCode       CHAR(10)     CONSTRAINT FacZipRequired NOT NULL,
  FacHireDate      DATE,
  FacDept          CHAR(6),
  CONSTRAINT PKFaculty PRIMARY KEY (FacSSN),
  CONSTRAINT UniqueFacName UNIQUE (FacFirstName, FacLastName) )

```

```

    FacRank           CHAR(4),
    FacSalary        DECIMAL(10,2),
    FacSupervisor    CHAR(11),
CONSTRAINT PKFaculty PRIMARY KEY (FacSSN),
CONSTRAINT FKFacSupervisor FOREIGN KEY (FacSupervisor) REFERENCES Faculty
    ON DELETE SET NULL
    ON UPDATE CASCADE )

CREATE TABLE Offering
( OfferNo          INTEGER,
  CourseNo         CHAR(6)      CONSTRAINT OffCourseNoRequired NOT NULL,
  OffLocation       VARCHAR(50),
  OffDays          CHAR(6),
  OffTerm          CHAR(6)      CONSTRAINT OffTermRequired NOT NULL,
  OffYear          INTEGER     CONSTRAINT OffYearRequired NOT NULL,
  FacSSN           CHAR(11),
  OffTime          DATE,
CONSTRAINT PKOffering PRIMARY KEY (OfferNo),
CONSTRAINT FKCourseNo FOREIGN KEY (CourseNo) REFERENCES Course
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
CONSTRAINT FKFacSSN FOREIGN KEY (FacSSN) REFERENCES Faculty
    ON DELETE SET NULL
    ON UPDATE CASCADE    )

CREATE TABLE Enrollment
( OfferNo          INTEGER,
  StdSSN           CHAR(11),
  EnrGrade         DECIMAL(3,2),
CONSTRAINT PKEnrollment PRIMARY KEY (OfferNo, StdSSN),
CONSTRAINT FKOfferNo FOREIGN KEY (OfferNo) REFERENCES Offering
    ON DELETE CASCADE
    ON UPDATE CASCADE,
CONSTRAINT FKStdSSN FOREIGN KEY (StdSSN) REFERENCES Student
    ON DELETE CASCADE
    ON UPDATE CASCADE    )

```

## Apéndice 3.B

### Resumen de sintaxis en SQL:2003

Este apéndice proporciona un resumen de la sintaxis de SQL:2003 para la sentencia CREATE TABLE, junto con otras sentencias relacionadas. Para ser breve, sólo se describe la sintaxis de las partes más comunes de las sentencias. SQL:2003 es la versión actual del SQL estándar. La sintaxis en SQL:2003 para las sentencias que se describen en este apéndice es idéntica a la sintaxis de los estándares previos de SQL, SQL:1999 y SQL-92. Para revisar la sintaxis completa,

consulte un libro de referencia sobre SQL:2003 o SQL-92, como Groff y Weinberg (2002). Las convenciones usadas en la notación de la sintaxis se listan antes de la sintaxis de la sentencia:

- Las palabras en mayúscula representan palabras reservadas.
- Las palabras con mayúsculas y minúsculas sin guiones representan nombres que el usuario sustituye.
- El asterisco \* después de un elemento de la sintaxis indica que se puede usar una lista separada por comas.
- El símbolo más + después de un elemento de la sintaxis indica que se puede usar una lista. No aparecen comas en la lista.
- Los nombres encerrados por los símbolos <> representan definiciones que se definen más adelante en la sintaxis. Las definiciones aparecen en una nueva línea con el elemento y los dos puntos seguidos por la sintaxis.
- Los corchetes [ ] encierran elementos opcionales.
- Las llaves { } encierran elementos alternativos. Se debe escoger un elemento de entre aquellos separados por las barras verticales |.
- Los paréntesis ( ) se representan a sí mismos.
- Los guiones dobles -- representan comentarios que no son parte de la sintaxis.

## Sintaxis CREATE TABLE<sup>11</sup>

```

CREATE TABLE TableName
( <Column-Definition>*    [ ,    <Table-Constraint>*  ]  )

<Column-Definition>: ColumnName DataType
[ DEFAULT { DefaultValue | USER | NULL } ]
[ <Embedded-Column-Constraint>+ ]

<Embedded-Column-Constraint>:
{ [ CONSTRAINT ConstraintName ] NOT NULL      |
  [ CONSTRAINT ConstraintName ] UNIQUE        |
  [ CONSTRAINT ConstraintName ] PRIMARY KEY   |
  [ CONSTRAINT ConstraintName ] FOREIGN KEY
    REFERENCES TableName [ (ColumnName) ]
  [ ON DELETE  <Action-Specification>  ]
  [ ON UPDATE  <Action-Specification>  ]  }

<Table-Constraint>: [ CONSTRAINT ConstraintName ]
{ <Primary-Key-Constraint>  |
  <Foreign-Key-Constraint>  |
  <Uniqueness-Constraint>  }

<Primary-Key-Constraint>: PRIMARY KEY (ColumnName*)
<Foreign-Key-Constraint>: FOREIGN KEY (ColumnName*)
  REFERENCES TableName [(ColumnName*)]
  [ ON DELETE  <Action-Specification>  ]
  [ ON UPDATE  <Action-Specification>  ]

```

<sup>11</sup> En el capítulo 14 se describe la restricción CHECK, un tipo importante de restricción de tabla.

```
<Action-Specification>: { CASCADE | SET NULL | SET DEFAULT | RESTRICT }

<Uniqueness-Constraint>: UNIQUE ( ColumnName* )
```

### Otras sentencias relacionadas

Las sentencias ALTER TABLE y DROP TABLE apoyan la modificación y eliminación de la definición de una tabla. La sentencia ALTER TABLE es particularmente útil, ya que las definiciones de las tablas cambian continuamente. En ambas sentencias, la palabra clave RESTRICT significa que la sentencia no se puede ejecutar si existen tablas relacionadas. La palabra clave CASCADE significa que la misma acción se ejecutará en las tablas relacionadas.

```
ALTER TABLE TableName
{ ADD { <Column-Definition> | <Table-Constraint> } |
  ALTER ColumnName { SET DEFAULT DefaultValue |
    DROP DEFAULT } |
  DROP ColumnName { CASCADE | RESTRICT } |
  DROP CONSTRAINT ConstraintName { CASCADE | RESTRICT } }

DROP TABLE TableName { CASCADE | RESTRICT }
```

### Notas sobre la sintaxis de Oracle

La sentencia CREATE TABLE en Oracle 10g SQL se apega mucho al estándar SQL:2003. La siguiente es una lista de las diferencias más significativas en la sintaxis:

- Oracle SQL no soporta la cláusula ON UPDATE para las restricciones de integridad referencial.
- Oracle SQL sólo soporta CASCADE y SET NULL como las especificaciones de las acciones para la cláusula ON DELETE. Si no se especifica una cláusula ON DELETE, no se permite la eliminación (restricción) en caso de que existan renglones relacionados.
- Oracle SQL no soporta la eliminación de columnas en la sentencia ALTER.
- Oracle SQL soporta la palabra clave MODIFY en lugar de ALTER en la sentencia ALTER TABLE (utilice MODIFYColumnName en lugar de ALTERColumnName).
- Oracle SQL soporta los cambios a los tipos de datos mediante la palabra clave MODIFY en la sentencia ALTER TABLE.

## Apéndice 3.C

### Generación de valores únicos para llaves primarias

El estándar SQL:2003 proporciona la cláusula GENERATED para soportar la generación de valores únicos para las columnas seleccionadas, generalmente llaves primarias. La cláusula GENERATED se usa en lugar del valor por omisión, tal como se muestra en la siguiente especificación de sintaxis. Comúnmente un tipo de dato de número entero como INTEGER debe usarse para columnas con la cláusula GENERATED. Las palabras clave START BY e INCREMENT BY se pueden usar para indicar el valor inicial y el valor de incremento. La palabra clave ALWAYS indica que el valor siempre se genera de forma automática. La cláusula BY DEFAULT le permite al usuario especificar un valor, sobreescribiendo la generación automática de valores.

```

<Column-Definition>: ColumnName DataType
[ <Default-Specification> ]
[ <Embedded-Column-Constraint>+ ]

<Default-Specification>:
{ DEFAULT { DefaultValue | USER | NULL } |
  GENERATED {ALWAYS | BY DEFAULT }-AS IDENTITY
  START WITH NumericConstant
  [ INCREMENT BY NumericConstant ] }

```

El cumplimiento de la sintaxis SQL:2003 para la cláusula GENERATED varía entre los DBMS. IBM DB2 se apega mucho a la sintaxis. Microsoft SQL Server usa una sintaxis ligeramente distinta y sólo soporta la opción ALWAYS cuando se usa también una sentencia SET IDENTITY. Microsoft Access proporciona el tipo de dato AutoNumber para generar valores únicos. Oracle utiliza objetos de secuencia en lugar de la cláusula GENERATED. Las secuencias de Oracle tienen características similares, con la excepción de que los usuarios deben mantener la asociación entre una secuencia y una columna, una carga no necesaria con el SQL:2003 estándar.

Los siguientes ejemplos contrastan los enfoques de SQL:2003 y Oracle para la generación automática de valores. Observe que la restricción de la llave primaria no se requiere para las columnas con valores generados, aunque los valores generados se usan en su mayoría para las llaves primarias. El ejemplo de Oracle incluye dos sentencias: una para la creación de la secuencia y otra para la creación de la tabla. Debido a que las secuencias no están asociadas con las columnas, Oracle proporciona funciones que deben usarse cuando se inserta una fila en una tabla. En contraste, el uso de funciones extra no es necesario en SQL:2003.

## Ejemplo de la cláusula GENERATED en SQL:2003

```

CREATE TABLE Customer
( CustNo INTEGER GENERATED ALWAYS AS IDENTITY
  START WITH 1 INCREMENT BY 1,
  ...,
  CONSTRAINT PKCustomer PRIMARY KEY (CustNo) )

```

## Ejemplo de secuencia de Oracle

```

CREATE SEQUENCE CustNoSeq START WITH 1 INCREMENT BY 1;

CREATE TABLE Customer
( CustNo INTEGER,
  ...,
  CONSTRAINT PKCustomer PRIMARY KEY (CustNo) );

```



# Capítulo

---

## Formulación de consultas con SQL

### Objetivos de aprendizaje

Este capítulo proporciona las bases para la formulación de consultas usando el estándar de la industria Structured Query Language (SQL). La formulación de consultas es el proceso de convertir la solicitud de datos en una sentencia del lenguaje de base de datos, como SQL. Después de este capítulo el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Escribir sentencias SQL SELECT para consultas con operadores de restricción, proyección y enlace.
- Utilizar las preguntas críticas para transformar la sentencia de un problema en una representación de base de datos.
- Escribir sentencias SELECT para enlaces difíciles que incluyen tres o más tablas, autoenlaces y múltiples enlaces entre tablas.
- Entender el propósito de la cláusula GROUP BY usando el proceso de evaluación conceptual.
- Escribir descripciones en español en sentencias documentadas de SQL.
- Escribir las sentencias INSERT, UPDATE y DELETE para modificar las filas de una tabla.

### Panorama general

---

El capítulo 3 proporcionó los fundamentos para usar las bases de datos relacionales. Pero lo más importante que usted aprendió fue acerca de las conexiones entre tablas y operadores fundamentales para extraer datos útiles. Este capítulo muestra cómo se puede aplicar este conocimiento usando las sentencias de manipulación de datos de SQL.

Gran parte de las habilidades con SQL u otros lenguajes de computación se derivan de la imitación de ejemplos. Este capítulo proporciona muchos ejemplos para facilitar el proceso de aprendizaje. Inicialmente se le presentan ejemplos relativamente sencillos para que se acople bien a la sentencia básica SELECT de SQL. Con el fin de prepararlo para ejemplos más complejos, se presentan dos guías de solución de problemas (proceso de evaluación conceptual y preguntas críticas). El proceso de evaluación conceptual explica el significado de la sentencia SELECT a través de la secuencia de operaciones y tablas intermedias que generan el resultado.

Las preguntas críticas le ayudan a transformar la sentencia de un problema en una representación relacional de base de datos en un lenguaje como SQL. Estas guías se usan para ayudarle a generar y explicar los problemas avanzados que se presentan en la última parte de este capítulo.

## 4.1 Antecedentes

---

Antes de utilizar SQL, es necesario entender su historia y alcance. Esta historia revela el origen del nombre y los esfuerzos para estandarizar el lenguaje. El alcance pone las múltiples partes de SQL en perspectiva. Usted ya ha revisado la sentencia CREATE TABLE en el capítulo 3. Las sentencias SELECT, UPDATE, DELETE e INSERT son los temas de este capítulo y del 9. Para ampliar su conocimiento debe estar enterado de otras partes de SQL y su uso en distintos contextos.

### 4.1.1 Breve historia de SQL

El lenguaje de consulta estructurada (SQL) tiene una peculiar historia. La tabla 4.1 describe los puntos clave en el desarrollo de SQL. SQL inició su vida como el lenguaje SQUARE en el proyecto del sistema R de IBM. El proyecto del sistema R fue una respuesta al interés generado por el doctor Ted Codd, quien escribió un documento en 1970 sobre bases de datos relacionales. El lenguaje SQUARE fue de alguna manera de naturaleza matemática. Después de conducir experimentos sobre los factores humanos, el equipo de investigaciones de IBM revisó el lenguaje y lo renombró SEQUEL (en lugar de SQUARE). Después de otra revisión el lenguaje fue titulado SEQUEL 2. Su nombre actual es SQL, resultado de los detalles legales que rodearon el nombre de SEQUEL. Dada esta historia de denominación, existe un número de profesionales de bases de datos, particularmente aquellos que trabajaron durante la década de 1970, que pronuncian el nombre como “sequel” en lugar de SQL.

En la actualidad SQL es un estándar internacional, aunque no siempre fue así.<sup>1</sup> Con la fuerza de IBM detrás de SQL, muchos imitadores usaron alguna variante de SQL. Así era el antiguo orden en la industria computacional cuando era dominada por IBM. Puede parecernos sorprendente, pero IBM no fue la primera compañía en comercializar SQL. Hasta antes del esfuerzo por desarrollar estándares durante la década de 1980, SQL estuvo en un estado de confusión. Muchos fabricantes implementaron diferentes subconjuntos de SQL con extensiones propias. El orden se restauró de algún modo gracias a los esfuerzos de estandarización del Instituto Nacional de Estándares de América (ANSI, por sus siglas en inglés), la Organización Internacional de Estándares (ISO) y la Comisión Electrónica Internacional (IEC). Aunque al inicio SQL no era el mejor lenguaje desarrollado para bases de datos, los esfuerzos en los avances de estándares mejoraron el lenguaje, así como la estandarización de sus especificaciones.

**TABLA 4.1**  
**Evolución de SQL**

Año	Evento
1972	Proyecto del sistema R de los laboratorios de investigación de IBM
1974	Desarrollo del lenguaje SQUARE
1975	Revisión del lenguaje y cambio de nombre a SEQUEL
1976	Revisión del lenguaje y cambio de nombre a SEQUEL 2
1977	Cambio de nombre a SQL
1978	Primera implementación comercial hecha por Oracle Corporation
1981	IBM produce SQL/DS con características de SQL
1986	Se aprueba el estándar SQL-86 (SQL1)
1989	Se aprueba el estándar SQL-89 (revisión de SQL-86)
1992	Se aprueba el estándar SQL-92 (SQL2)
1999	Se aprueba el estándar SQL:1999 (SQL3)
2003	Aprobación de SQL:2003

<sup>1</sup> El doctor Michael Stonebraker, pionero de las bases de datos, se ha referido a SQL como “lenguaje intergaláctico de datos”.

El tamaño y alcance del estándar SQL ha aumentado de forma significativa desde la aprobación del primer estándar. El estándar original (SQL-86) contenía aproximadamente 150 páginas, mientras que el estándar SQL-92 contiene más de 600 páginas. En contraste, los estándares más recientes (SQL:1999 y SQL:2003) contienen más de 2 000 páginas. Los estándares iniciales (SQL-86 y SQL-89) tenían dos niveles (básico y completo). SQL-92 añadió un tercer nivel (básico, intermedio y completo). Los estándares SQL:1999 y SQL:2003 contienen un nivel llamado Core SQL en conjunto con las partes y paquetes para las características fuera del núcleo. SQL:2003 contiene tres partes nucleares, seis partes opcionales y siete paquetes opcionales.

La debilidad de los estándares de SQL es la ausencia de pruebas de cumplimiento. Desde 1996, el Departamento de Comercio del Instituto Nacional de Estándares y Tecnología de Estados Unidos realizó pruebas de cumplimiento para asegurarse de que el software gubernamental pudiera trasladarse de un DBMS a otro. Sin embargo, desde 1996, las reclamaciones de los vendedores de DBMS se han sustituido por pruebas independientes de conformidad. Incluso para el Core SQL, la mayoría de los vendedores carecieron de soporte para algunos puntos y proporcionaron soporte propietario para otros. La conformidad ha tenido grandes variaciones con las partes opcionales y paquetes. Para escribir código portátil para SQL se requiere de un estudio minucioso del Core SQL, pero no es posible para las partes avanzadas de SQL.

La presentación de este capítulo se limita a un subconjunto del Core SQL:2003. La mayoría de las características presentadas en este capítulo eran parte de SQL-92 así como del Core SQL:2003. En otros capítulos se presentan otras partes de SQL, al igual que importantes características de los paquetes seleccionados de SQL:2003.

#### 4.1.2 Alcance de SQL

SQL fue diseñado como un lenguaje para la definición, manipulación y control de bases de datos. La tabla 4.2 muestra un breve resumen de las sentencias más importantes de SQL. Sólo los administradores de bases de datos usan la mayoría de las sentencias de definición y control de las bases de datos. Ya ha visto la sentencia CREATE TABLE en el capítulo 3. Este capítulo y el capítulo 9 abarcan las sentencias de manipulación de la base de datos. Los usuarios avanzados y los analistas usan las sentencias de manipulación de base de datos. El capítulo 10 abarca la sentencia CREATE VIEW. Esta sentencia puede ser usada tanto por el administrador como por el analista. El capítulo 11 incluye la sentencia de CREATE TRIGGER usada tanto por el administrador como por el analista de base de datos. El capítulo 14 abarca las sentencias GRANT, REVOKE y CREATE ASSERTION usadas principalmente por los administradores de bases de datos. Las sentencias de control de transacciones (COMMIT y ROLLBACK), que se presentan en el capítulo 15, son utilizadas por el analista.

SQL puede utilizarse en dos contextos: individual y embebido (incrustado). En el contexto individual el usuario ejecuta las sentencias SQL mediante un editor especial. El editor avisa al usuario en caso de que existan errores de sintaxis y envía las sentencias al DBMS. La presentación en este capítulo asume el uso individual. En el contexto embebido, un programa en ejecución envía sentencias SQL y el DBMS envía los resultados de regreso al programa. El programa incluye las sentencias SQL en conjunto con las sentencias del lenguaje anfitrión de

#### **contextos de uso de SQL**

las sentencias SQL pueden utilizarse de forma individual con un editor especializado, o embebidas dentro de un programa computacional.

**TABLA 4.2** Sentencias seleccionadas de SQL

Tipo de sentencia	Sentencias	Propósito
<i>Definición de bases de datos</i>	CREATE SCHEMA, TABLE, VIEW ALTER TABLE	Definir una nueva base de datos, tabla y vista Modificar la definición de una tabla
<i>Manipulación de bases de datos</i>	SELECT UPDATE, DELETE, INSERT	Extraer los contenidos de las tablas Modificar, eliminar y agregar filas
<i>Control de bases de datos</i>	COMMIT, ROLLBACK GRANT, REVOKE CREATE ASSERTION CREATE TRIGGER	Terminar y deshacer transacciones Agregar y eliminar derechos de acceso Definir una restricción de integridad Definir una regla de base de datos

programación, tales como Java o Visual Basic. Las sentencias adicionales permiten que sentencias SQL (como SELECT) puedan utilizarse dentro de los programas computacionales. El capítulo 11 abarca el SQL embbebido.

## 4.2 Empecemos con la sentencia SELECT

---

La sentencia SELECT soporta la recuperación de datos de una o más tablas. Esta sección describe un formato simplificado de la sentencia SELECT. Los formatos más complejos se presentan en el capítulo 9. La sentencia SELECT descrita aquí tiene el siguiente formato:

```
SELECT <list of columns and expressions usually involving columns>
      FROM <list of tables and join operations>
      WHERE <list of row conditions connected by AND, OR, NOT>
      GROUP BY <list of grouping columns>
      HAVING <list of group conditions connected by AND, OR, NOT>
      ORDER BY <list of sorting specifications>
```

En el formato anterior, las palabras en mayúscula son palabras clave. Usted reemplazaría los pico paréntesis <> con información para que la sentencia tenga significado. Por ejemplo, después de la palabra clave SELECT, teclee la lista de columnas que debe aparecer en el resultado, y no teclee los pico paréntesis. El resultado que se listará puede contener columnas tales como *StdFirstName* o expresiones que incluyen constantes, nombres de columnas y funciones. Las expresiones de ejemplo son *Price \* Qty* y *1.1 \* FacSalary*. Para crear nombres que tengan sentido para las columnas a imprimir, se puede renombrar la columna en la tabla resultante usando la palabra clave AS. Por ejemplo, *SELECT Price \* Qty AS Amount* renombra la expresión *Price \* Qty* por *Amount* en la tabla resultante.

Este capítulo presenta numerosos ejemplos para describir el formato de SELECT y mostrar el significado de las sentencias. Los ejemplos se proporcionan tanto para Microsoft Access, un popular DBMS de escritorio, y Oracle, un DBMS corporativo muy importante. La mayoría de los ejemplos se ejecutan en ambos sistemas. A menos que así se indique, los ejemplos corren para las versiones 1997 a 2003 de Access, y 8i a 10g de Oracle. Se indican los ejemplos que sólo se ejecutan en un producto. Además de los ejemplos, el apéndice 4.B resume las diferencias de sintaxis entre los principales DBMS.

Los ejemplos usan las tablas de la base de datos de la universidad proporcionadas en el capítulo 3. Los contenidos de las tablas se listan desde la tabla 4.3 hasta la 4.7. Las sentencias CREATE TABLE se enlistan en el apéndice 3.A. Para su referencia, el diagrama de relación que

### expresión

una combinación de constantes, columnas, nombres, funciones y operadores que generan un valor. En las condiciones y los resultados de las columnas, las expresiones se pueden utilizar en cualquier lugar donde aparezca el nombre de una columna.

**TABLA 4.3 Ejemplo de la tabla Student**

StdSSN	StdFirstName	StdLastName	StdCity	StdState	StdZip	StdMajor	StdClass	StdGPA
123-45-6789	HOMER	WELLS	SEATTLE	WA	98121-1111	IS	FR	3.00
124-56-7890	BOB	NORBERT	BOTHELL	WA	98011-2121	FIN	JR	2.70
234-56-7890	CANDY	KENDALL	TACOMA	WA	99042-3321	ACCT	JR	3.50
345-67-8901	WALLY	KENDALL	SEATTLE	WA	98123-1141	IS	SR	2.80
456-78-9012	JOE	ESTRADA	SEATTLE	WA	98121-2333	FIN	SR	3.20
567-89-0123	MARIAH	DODGE	SEATTLE	WA	98114-0021	IS	JR	3.60
678-90-1234	TESS	DODGE	REDMOND	WA	98116-2344	ACCT	SO	3.30
789-01-2345	ROBERTO	MORALES	SEATTLE	WA	98121-2212	FIN	JR	2.50
876-54-3210	CRISTOPHER	COLAN	SEATTLE	WA	98114-1332	IS	SR	4.00
890-12-3456	LUKE	BRAZZI	SEATTLE	WA	98116-0021	IS	SR	2.20
901-23-4567	WILLIAM	PILGRIM	BOTHELL	WA	98113-1885	IS	SO	3.80

**TABLA 4.4A** Ejemplo de la tabla *Faculty* (primera parte)

FacSSN	FacFirstName	FacLastName	FacCity	FacState	FacDept	FacRank	FacSalary
098-76-5432	LEONARD	VINCE	SEATTLE	WA	MS	ASST	\$35,000
543-21-0987	VICTORIA	EMMANUEL	BOTHELL	WA	MS	PROF	\$120,000
654-32-1098	LEONARD	FIBON	SEATTLE	WA	MS	ASSC	\$70,000
765-43-2109	NICKI	MACON	BELLEVUE	WA	FIN	PROF	\$65,000
876-54-3210	CRISTOPHER	COLAN	SEATTLE	WA	MS	ASST	\$40,000
987-65-4321	JULIA	MILLS	SEATTLE	WA	FIN	ASSC	\$75,000

**TABLA 4.4B**  
Ejemplo de la tabla  
*Faculty* (segunda  
parte)

FacSSN	FacSupervisor	FacHireDate	FacZipCode
098-76-5432	654-32-1098	10-Apr-1995	98111-9921
543-21-0987		15-Apr-1996	98011-2242
654-32-1098	543-21-0987	01-May-1994	98121-0094
765-43-2109		11-Apr-1997	98015-9945
876-54-3210	654-32-1098	01-Mar-1999	98114-1332
987-65-4321	765-43-2109	15-Mar-2000	98114-9954

**TABLA 4.5** Ejemplo de la tabla *Offering*

OfferNo	CourseNo	OffTerm	OffYear	OffLocation	OffTime	FacSSN	OffDays
1111	IS320	SUMMER	2006	BLM302	10:30 AM		MW
1234	IS320	FALL	2005	BLM302	10:30 AM	098-76-5432	MW
2222	IS460	SUMMER	2005	BLM412	1:30 PM		TTH
3333	IS320	SPRING	2006	BLM214	8:30 AM	098-76-5432	MW
4321	IS320	FALL	2005	BLM214	3:30 PM	098-76-5432	TTH
4444	IS320	WINTER	2006	BLM302	3:30 PM	543-21-0987	TTH
5555	FIN300	WINTER	2006	BLM207	8:30 AM	765-43-2109	MW
5678	IS480	WINTER	2006	BLM302	10:30 AM	987-65-4321	MW
5679	IS480	SPRING	2006	BLM412	3:30 PM	876-54-3210	TTH
6666	FIN450	WINTER	2006	BLM212	10:30 AM	987-65-4321	TTH
7777	FIN480	SPRING	2006	BLM305	1:30 PM	765-43-2109	MW
8888	IS320	SUMMER	2006	BLM405	1:30 PM	654-32-1098	MW
9876	IS460	SPRING	2006	BLM307	1:30 PM	654-32-1098	TTH

**TABLA 4.6**  
Ejemplo de la tabla  
*Course*

CourseNo	CrsDesc	CrsUnits
FIN300	FUNDAMENTOS DE FINANZAS	4
FIN450	PRINCIPIOS DE INVERSIÓN	4
FIN480	FINANZAS CORPORATIVAS	4
IS320	FUNDAMENTOS DE PROGRAMACIÓN DE NEGOCIOS	4
IS460	ANÁLISIS DE SISTEMAS	4
IS470	COMUNICACIÓN DE DATOS DE NEGOCIOS	4
IS480	FUNDAMENTOS DE ADMINISTRACIÓN DE BASES DE DATOS	4

**TABLA 4.7**  
**Ejemplo de la tabla**  
*Enrollment*

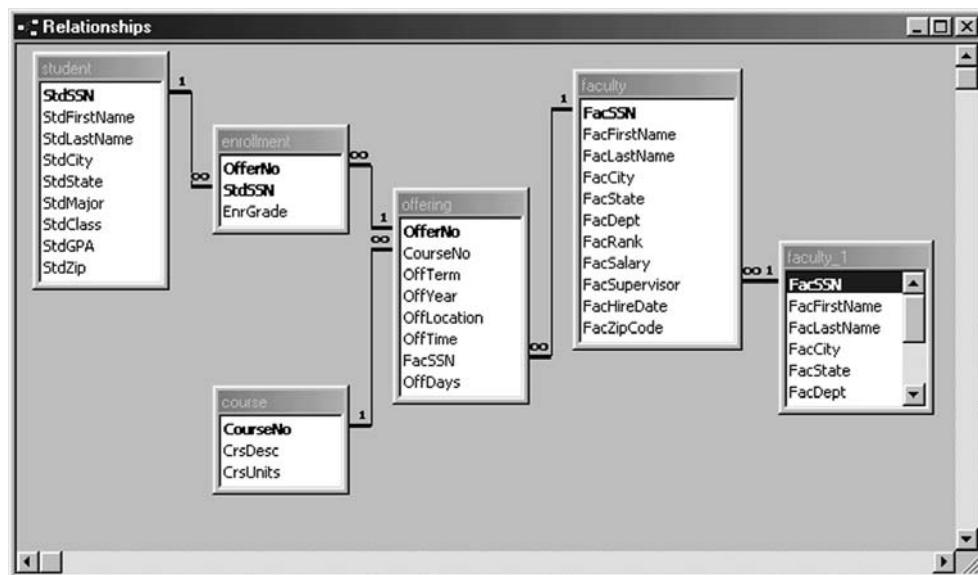
OfferNo	StdSSN	EnrGrade
1234	123-45-6789	3.3
1234	234-56-7890	3.5
1234	345-67-8901	3.2
1234	456-78-9012	3.1
1234	567-89-0123	3.8
1234	678-90-1234	3.4
4321	123-45-6789	3.5
4321	124-56-7890	3.2
4321	789-01-2345	3.5
4321	876-54-3210	3.1
4321	890-12-3456	3.4
4321	901-23-4567	3.1
5555	123-45-6789	3.2
5555	124-56-7890	2.7
5678	123-45-6789	3.2
5678	234-56-7890	2.8
5678	345-67-8901	3.3
5678	456-78-9012	3.4
5678	567-89-0123	2.6
5679	123-45-6789	2
5679	124-56-7890	3.7
5679	678-90-1234	3.3
5679	789-01-2345	3.8
5679	890-12-3456	2.9
5679	901-23-4567	3.1
6666	234-56-7890	3.1
6666	567-89-0123	3.6
7777	876-54-3210	3.4
7777	890-12-3456	3.7
7777	901-23-4567	3.4
9876	124-56-7890	3.5
9876	234-56-7890	3.2
9876	345-67-8901	3.2
9876	456-78-9012	3.4
9876	567-89-0123	2.6
9876	678-90-1234	3.3
9876	901-23-4567	4

muestra la llave primaria y foránea se repite en la figura 4.1. Recordemos que la tabla *FacultyI* en relación con la tabla *Faculty* representa una referencia a sí misma con *FacSupervisor* como llave foránea.

#### 4.2.1 Problemas de tabla simple

Empecemos con la sencilla sentencia SELECT del ejemplo 4.1. En todos los ejemplos las palabras clave aparecen en mayúscula, mientras que la información específica de la consulta aparece tanto en mayúsculas como en minúsculas. En el ejemplo 4.1, sólo se lista la tabla *Student* de la cláusula FROM, ya que las condiciones de la cláusula WHERE y las columnas después de la palabra clave SELECT involucran solamente la tabla *Student*. En Oracle, un punto y coma o / (en una línea aparte) finaliza una sentencia.

**FIGURA 4.1**  
Ventana de relaciones para la base de datos de la universidad



**TABLA 4.8**  
Operadores estándares de comparación

Operador de comparación	Significado
=	igual a
<	menor que
>	mayor que
<=	menor o igual a
>=	mayor o igual a
<> o !=	diferente (revise su DBMS)

#### EJEMPLO 4.1

#### Prueba de filas usando la cláusula WHERE

Extraiga el nombre, ciudad y promedio del curso de los estudiantes con promedio alto (mayor o igual a 3.7). El resultado se muestra después de la sentencia SELECT.

```
SELECT StdFirstName, StdLastName, StdCity, StdGPA
  FROM Student
 WHERE StdGPA >= 3.7
```

StdFirstName	StdLastName	StdCity	StdGPA
CRISTOPHER	COLAN	SEATTLE	4.00
WILLIAM	PILGRIM	BOTHELL	3.80

La tabla 4.8 ilustra los operadores estándar de comparación. Tome en cuenta que el símbolo de algunos operadores depende del DBMS.

El ejemplo 4.2 es más simple que el ejemplo 4.1. El resultado es idéntico a la tabla original *Faculty* de la tabla 4.4. El ejemplo 4.2 utiliza un atajo para listar todas las columnas. El asterisco \* en la lista de las columnas indica que todas las columnas de las tablas de la cláusula FROM aparecen dentro del resultado. El asterisco sirve como el carácter comodín que concuerda con todos los nombres de las columnas.

**EJEMPLO 4.2****Muestre todas las columnas**

Liste todas las columnas y filas de la tabla *Faculty*. La tabla resultante se muestra en dos partes.

```
SELECT * FROM Faculty
```

FacSSN	FacFirstName	FacLastName	FacCity	FacState	FacDept	FacRank	FacSalary
098-76-5432	LEONARD	VINCE	SEATTLE	WA	MS	ASST	\$35,000
543-21-0987	VICTORIA	EMMANUEL	BOTHELL	WA	MS	PROF	\$120,000
654-32-1098	LEONARD	FIBON	SEATTLE	WA	MS	ASSC	\$70,000
765-43-2109	NICKI	MACON	BELLEVUE	WA	FIN	PROF	\$65,000
876-54-3210	CRISTOPHER	COLAN	SEATTLE	WA	MS	ASST	\$40,000
987-65-4321	JULIA	MILLS	SEATTLE	WA	FIN	ASSC	\$75,000

FacSSN	FacSupervisor	FacHireDate	FacZipCode
098-76-5432	654-32-1098	10-Apr-1995	98111-9921
543-21-0987		15-Apr-1996	98011-2242
654-32-1098	543-21-0987	01-May-1994	98121-0094
765-43-2109		11-Apr-1997	98015-9945
876-54-3210	654-32-1098	01-Mar-1999	98114-1332
987-65-4321	765-43-2109	15-Mar-2000	98114-9954

El ejemplo 4.3 ilustra las expresiones en las cláusulas SELECT y WHERE. La expresión dentro de la cláusula SELECT incrementa el salario en 10 por ciento. La palabra clave AS se usa para renombrar la columna calculada. Sin el nombre, la mayoría de los DBMS generaría un nombre sin sentido, tal como Expr001. La expresión de la cláusula WHERE extrae el año de la fecha de contratación. Como las funciones para el tipo de dato fecha no son estándares, se proporcionan las formulaciones de Access y Oracle. Para hacer más eficiente SQL con un DBMS en particular, usted necesitará estudiar las funciones disponibles, especialmente las de las columnas de fechas.

**EJEMPLO 4.3  
(Access)****Expresiones de las cláusulas SELECT y WHERE**

Liste el nombre, la ciudad y el incremento salarial de las contrataciones de la facultad después de 1996. La función **año** extrae la parte del año de una columna con un tipo de dato de fecha.

```
SELECT FacFirstName, FacLastName, FacCity,
       FacSalary*1.1 AS IncreasedSalary, FacHireDate
  FROM Faculty
 WHERE year(FacHireDate) > 1996
```

FacFirstName	FacLastName	FacCity	IncreasedSalary	FacHireDate
NICKI	MACON	BELLEVUE	71500	11-Apr-1997
CRISTOPHER	COLAN	SEATTLE	44000	01-Mar-1999
JULIA	MILLS	SEATTLE	82500	15-Mar-2000

**EJEMPLO 4.3  
(Oracle)****Expresiones de las cláusulas SELECT y WHERE**

La función **to\_char** extrae los cuatro dígitos del año de la columna *FacHireDate* y la función **to\_number** convierte la representación de caracteres del año en un número.

```
SELECT FacFirstName, FacLastName, FacCity,
       FacSalary*1.1 AS IncreasedSalary, FacHireDate
  FROM Faculty
 WHERE to_number(to_char(FacHireDate, 'YYYY')) > 1996
```

La coincidencia inexacta mantiene las similitudes que coinciden en algún patrón en lugar de coincidir en una cadena de caracteres idénticos. Uno de los tipos más comunes de coincidencia inexacta es encontrar valores que tengan un prefijo común, tal como “IS4” (nivel 400 de cursos de IS). El ejemplo 4.4 utiliza el operador LIKE en conjunto con un patrón de caracteres de coincidencia \* para realizar la coincidencia de prefijos.<sup>2</sup> La cadena de caracteres constante ‘IS4\*’ significa que la cadena de coincidencia empieza con “IS4” y termina con cualquier cosa. El carácter comodín \* coincide con cualquier cadena de caracteres. La formulación de Oracle del ejemplo 4.4 usa el símbolo de porcentaje %, estándar de SQL:2003 para el carácter comodín. Tome en cuenta que los caracteres constantes deben encerrarse entre comillas.<sup>3</sup>

**EJEMPLO 4.4  
(Access)****Coincidencia inexacta con el operador LIKE**

Lista del nivel avanzado de cursos de IS.

```
SELECT *
  FROM Course
 WHERE CourseNo LIKE 'IS4*''
```

CourseNo	CrsDesc	CrsUnits
IS460	ANÁLISIS DE SISTEMAS	4
IS470	COMUNICACIÓN DE DATOS DE NEGOCIOS	4
IS480	FUNDAMENTOS DE ADMINISTRACIÓN DE BASES DE DATOS	4

**EJEMPLO 4.4  
(Oracle)****Coincidencia inexacta con el operador LIKE**

Lista del nivel avanzado de cursos de IS.

```
SELECT *
  FROM Course
 WHERE CourseNo LIKE 'IS4%''
```

<sup>2</sup> Iniciando con Access 2002, los caracteres de búsqueda de patrones de SQL:2003 se pueden usar especificando el modo de consultas ANSI 92 en la ventana de opciones. Como las primeras versiones de Access no soportan esta opción y no está por omisión en Access 2002, este libro usa los caracteres de búsqueda de patrones \* y ? para las sentencias SQL de Access.

<sup>3</sup> La mayoría de los DBMS requieren comillas simples, estándar de SQL:2003. Microsoft Access permite comillas simples o dobles para las constantes de cadenas de caracteres.

Otro tipo común de coincidencia inexacta es la combinación de cadenas que contienen subcadenas. Para realizar este tipo de búsqueda se debe usar un carácter comodín antes y después de la subcadena. Por ejemplo, para encontrar los cursos que contienen la palabra DATABASE en cualquier parte de la descripción del curso, escriba la condición: *CrsDesc LIKE '\*DATABASE\** en Access. En Oracle sería *CrsDesc LIKE '%DATABASE%'*.

El carácter comodín no es el único de búsqueda de patrones. SQL:2003 especifica el guión bajo \_ para búsquedas de cualquier carácter. Algunos DBMS, como Access, usan el signo de interrogación ? para la búsqueda de cualquier carácter. Además, la mayoría de los DBMS contienen caracteres de búsqueda de patrones para buscar un rango de caracteres (por ejemplo, los dígitos del 0 al 9) y cualquier carácter de la lista de caracteres. Los símbolos utilizados para estos patrones de búsqueda de caracteres no son estándares. Para ser un experto en las condiciones búsquedas de patrones inexactas, deberá estudiar los caracteres de búsqueda de patrones disponibles en su DBMS.

Además, para realizar la búsqueda de patrones con cadenas de caracteres, puede utilizar búsqueda exacta con el operador de comparación de igualdad =. Por ejemplo, la condición *CourseNo = 'IS480'* coincide con una fila de la tabla *Course*. Para los patrones de búsquedas exactas e inexactas son importantes las mayúsculas y las minúsculas. Algunos DBMS, como Microsoft Access, no son sensibles a mayúsculas y minúsculas. En Access SQL, las condiciones previas coinciden con "is480", "Is480" e "iS480" además de "IS480". Otros DBMS, como Oracle, son sensibles a mayúsculas y minúsculas. En Oracle SQL, la condición previa sólo coincide con "IS480", y no con "is480", "Is480" o "iS480". Para aligerar la confusión usted puede utilizar las funciones **upper** o **lower** de Oracle para convertir la cadena de caracteres en mayúsculas o minúsculas, respectivamente.

El ejemplo 4.5 ilustra el rango de coincidencia de una columna con el tipo de datos fecha. En Access SQL, el símbolo de número encierra las constantes de las fechas; mientras que en Oracle SQL, las comillas sencillas encierran las constantes de las fechas. Las columnas de tipo fecha pueden compararse como números con los operadores generales de comparación (=, <, etc.). El operador BETWEEN-AND define un intervalo cerrado (incluye los extremos). En el ejemplo 4.5 de Access la condición BETWEEN-AND es un atajo para *FacHireDate >=#1/1/1999# AND FacHireDate <= #12/31/2000#*.

#### **operador**

##### **BETWEEN-AND**

un operador para comparar una columna numérica o de tipo fecha contra un rango de valores. El operador BETWEEN-AND regresa verdadero si la columna es mayor o igual que el primer valor y menor o igual que el segundo valor.

#### **EJEMPLO 4.5 (Access)**

##### **Condiciones para las columnas de fecha**

Liste el nombre y la fecha de contratación de los profesores contratados en 1999 o 2000.

```
SELECT FacFirstName, FacLastName, FacHireDate
  FROM Faculty
 WHERE FacHireDate BETWEEN #1/1/1999# AND #12/31/2000#
```

FacFirstName	FacLastName	FacHireDate
CRISTOPHER	COLAN	01-Mar-1994
JULIA	MILLS	15-Mar-2000

#### **EJEMPLO 4.5 (Oracle)**

##### **Condiciones para las columnas de fecha**

En Oracle SQL, el formato estándar para las fechas es DD-Mon-YYYY, donde DD es el número de día, Mon es la abreviación del mes y YYYY es el año en cuatro dígitos.

```
SELECT FacFirstName, FacLastName, FacHireDate
  FROM Faculty
 WHERE FacHireDate BETWEEN '1-Jan-1999' AND '31-Dec-2000'
```

Además de comparar las columnas con valores específicos, algunas veces requerirá compararlas con la ausencia de valor. Los valores nulos se utilizan cuando la columna no tiene un valor normal. Un nulo puede significar que el valor es desconocido o que el valor no se requiere en la fila. Para la tabla *Offering* un valor nulo de *FacSSN* significa que aún no se ha asignado al instructor. La comparación de valores nulos se hace con el operador de comparación IS NULL, tal como se muestra en el ejemplo 4.6. También puede probar para un valor normal utilizando IS NOT NULL.

**EJEMPLO 4.6****Comparación de valores nulos**

Enliste el número de cursos ofrecidos y el número de cursos del verano 2006 ofrecidos sin instructor asignado.

```
SELECT OfferNo, CourseNo
  FROM Offering
 WHERE FacSSN IS NULL AND OffTerm = 'SUMMER'
   AND OffYear = 2006
```

OfferNo	CourseNo
1111	IS320

**combinación AND y OR**

siempre use paréntesis para realizar la agrupación de las condiciones explícitas.

El ejemplo 4.7 ilustra una expresión lógica compleja que incluye los operadores lógicos AND y OR. Cuando se combinan AND y OR en una expresión lógica es buena idea el uso de paréntesis. De lo contrario, el lector de la sentencia SELECT quizás no entienda cómo se agrupan las condiciones AND y OR. Sin los paréntesis, usted dependerá de la forma inicial en que se agrupan las condiciones AND y OR.

**EJEMPLO 4.7****Expresión lógica compleja**

Enliste el número de cursos ofrecidos, el número de curso y el número de seguridad social del profesor para los cursos agendados en otoño de 2005 o invierno de 2006.

```
SELECT OfferNo, CourseNo, FacSSN
  FROM Offering
 WHERE (OffTerm = 'FALL' AND OffYear = 2005)
   OR (OffTerm = 'WINTER' AND OffYear = 2006)
```

OfferNo	CourseNo	FacSSN
1234	IS320	098-76-5432
4321	IS320	098-76-5432
4444	IS320	543-21-0987
5555	FIN300	765-43-2109
5678	IS480	987-65-4321
6666	FIN450	987-65-4321

**4.2.2 Tablas enlazadas (*joining*)**

El ejemplo 4.8 muestra el enlace (*join*) de las tablas *Course* y *Offering*. Las condiciones del enlace *Course.CourseNo = Offering.CourseNo* se especifican en la cláusula WHERE.

**EJEMPLO 4.8  
(Access)****Enlace de tablas mostrando sólo las columnas de una tabla**

Liste el número de cursos ofrecidos, el número de curso, los días y el tiempo de los cursos ofrecidos que contengan las palabras *database* o *programming* en la descripción del curso e impartidos en la primavera de 2006. La versión de Oracle de este ejemplo usa % en lugar de \* como carácter comodín.

```
SELECT OfferNo, Offering.CourseNo, OffDays, OffTime
  FROM Offering, Course
 WHERE OffTerm = 'SPRING' AND OffYear = 2006
   AND (CrsDesc LIKE '*DATABASE*'
        OR CrsDesc LIKE '*PROGRAMMING*')
   AND Course.CourseNo = Offering.CourseNo
```

OfferNo	CourseNo	OffDays	OffTime
3333	IS320	MW	8:30 AM
5679	IS480	TTH	3:30 PM

Existen dos puntos de interés adicional en el ejemplo 4.8. Primero, la columna de nombre *CourseNo* debe ser *calificada* (con el prefijo) con un nombre de tabla (*Course* u *Offering*). De lo contrario, la sentencia SELECT es ambigua ya que *CourseNo* puede hacer referencia a la tabla *Course* u *Offering*. Segundo, se deben listar ambas tablas en la cláusula FROM aunque las columnas del resultado provengan solamente de la tabla *Offering*. La tabla *Course* es necesaria en la cláusula FROM ya que las condiciones de la cláusula WHERE hacen referencia a *CrsDesc*, una columna de la tabla *Course*.

El ejemplo 4.9 demuestra otro enlace, pero esta vez las columnas resultantes provienen de ambas tablas. Existen condiciones en cada tabla además de las de enlace. La sintaxis de Oracle utiliza el % en lugar del \* como carácter comodín.

**EJEMPLO 4.9  
(Access)****Enlace las tablas y muestre las columnas de ambas tablas**

Liste el número del curso ofrecido, el número de curso y el nombre del profesor de los cursos de IS que se ofrecen durante el otoño de 2005 impartidos por los profesores asistentes.

```
SELECT OfferNo, CourseNo, FacFirstName, FacLastName
  FROM Offering, Faculty
 WHERE OffTerm = 'FALL' AND OffYear = 2005
   AND FacRank = 'ASST' AND CourseNo LIKE 'IS*'
   AND Faculty.FacSSN = Offering.FacSSN
```

OfferNo	CourseNo	FacFirstName	FacLastName
1234	IS320	LEONARD	VINCE
4321	IS320	LEONARD	VINCE

**EJEMPLO 4.9  
(Oracle)****Enlace las tablas y muestre las columnas de ambas tablas**

Liste el número de oferta, el número de curso y el nombre del profesor de los cursos de IS que se ofrecen durante el otoño de 2005 impartidos por los profesores asistentes.

```
SELECT OfferNo, CourseNo, FacFirstName, FacLastName
  FROM Offering, Faculty
 WHERE OffTerm = 'FALL' AND OffYear = 2005
   AND FacRank = 'ASST' AND CourseNo LIKE 'IS%'
   AND Faculty.FacSSN = Offering.FacSSN
```

En el estándar SQL:2003, el operador de enlace puede expresarse de forma directa en la cláusula FROM en lugar de expresarlo en las cláusulas FROM y WHERE, tal como se muestra en los ejemplos 4.8 y 4.9. Tome en cuenta que Oracle desde la versión 9i soporta los operadores de enlace con la cláusula FROM, pero en las versiones previas no se soporta el operador de enlace con la cláusula FROM. Para hacer una operación de enlace con la cláusula FROM, use las palabras clave INNER JOIN como se muestra en el ejemplo 4.10. Las condiciones de enlace se indican por la palabra clave ON dentro de la cláusula FROM. Note que la condición de enlace no aparece más en la cláusula WHERE.

**EJEMPLO 4.10  
(Access)****Enlace de tablas con el uso de la operación de enlace en la cláusula FROM**

Liste el número de oferta, el número de curso y el nombre del profesor de los cursos de IS que se ofrecen durante el otoño de 2005 impartidos por los profesores asistentes (el resultado es idéntico al ejemplo 4.9). En Oracle debe utilizar % en lugar de \*.

```
SELECT OfferNo, CourseNo, FacFirstName, FacLastName
  FROM Offering INNER JOIN Faculty
    ON Faculty.FacSSN = Offering.FacSSN
 WHERE OffTerm = 'FALL' AND OffYear = 2005
   AND FacRank = 'ASST' AND CourseNo LIKE 'IS*''
```

**recordatorio**  
**GROUP BY**  
las columnas en la cláusula SELECT pueden estar en la cláusula GROUP BY o ser parte del resumen de un cálculo con una función agregada.

**4.2.3 Resumen de tablas con GROUP BY y HAVING**

Hasta ahora, los resultados de todos los ejemplos en esta sección se relacionan con filas individuales. Incluso el ejemplo 4.9 relaciona una combinación de columnas de las filas *Offering* y *Faculty*. Como se menciona en el capítulo 3, algunas veces es importante mostrar las filas agrupadas. Las cláusulas GROUP BY y HAVING se usan para mostrar los resultados de grupos de filas en lugar de filas individuales.

El ejemplo 4.11 ilustra la cláusula GROUP BY para resumir grupos de filas. Cada fila resultante contiene el valor de la columna de agrupación (*StdMajor*) junto con el cálculo agregado de resumen de filas con el mismo valor para la columna de agrupación. La cláusula GROUP BY debe contener cada columna de la cláusula SELECT, excepto las expresiones agregadas. Por ejemplo, al agregar la columna *StdClass* en la cláusula SELECT se invalidará el ejemplo 4.11 hasta que se agregue *StdClass* a la cláusula GROUP BY.

**EJEMPLO 4.11****Agrupación en una sola columna**

Resumen del promedio de los estudiantes por carrera.

```
SELECT StdMajor, AVG(StdGPA) AS AvgGPA
  FROM Student
 GROUP BY StdMajor
```

StdMajor	AvgGPA
ACCT	3.39999997615814
FIN	2.80000003178914
IS	3.23333330949148

**uso de la función****COUNT**

COUNT(\*) y COUNT(column) generan resultados idénticos excepto cuando la “columna” contiene valores nulos. Vea el capítulo 9 para obtener más detalles acerca del efecto de los valores nulos en las funciones agregadas.

La tabla 4.9 muestra las funciones agregadas estándar. Si existe un cálculo estadístico que no puede ser realizado con estas funciones, revise su DBMS. La mayoría de los DBMS tienen muchas funciones además de las estándares.

Las funciones COUNT, AVG y SUM soportan la palabra clave DISTINCT para restringir el cálculo para obtener columnas con valores únicos. El ejemplo 4.12 muestra la palabra clave DISTINCT para la función COUNT. Este ejemplo extrae el número de cursos ofrecidos en un año, así como el número de cursos distintos impartidos. Algunos DBMS como Microsoft Access

**EJEMPLO 4.12**  
**(Oracle)****Conteo de filas y valores únicos en las columnas**

Resumen del número de cursos ofrecidos y de los cursos únicos por año.

```
SELECT OffYear, COUNT(*) AS NumOfferings,
       COUNT(DISTINCT CourseNo) AS NumCourses
  FROM Offering
 GROUP BY OffYear
```

OffYear	NumOfferings	NumCourses
2005	3	2
2006	10	6

**TABLA 4.9**  
Funciones agregadas estándar

Función agregada	Significado y comentarios
COUNT(*)	Calcula el número de filas.
COUNT(column)	Cuenta los valores no nulos de las columnas; se puede usar DISTINCT para contar las columnas de valores únicos.
AVG	Calcula el promedio de una columna numérica o de una expresión excluyendo los valores nulos; se puede utilizar DISTINCT para calcular el promedio de las columnas de valores únicos.
SUM	Calcula la suma del valor numérico de la columna o de una expresión excluyendo los valores nulos; se puede utilizar DISTINCT para calcular el promedio de las columnas de valores únicos.
MIN	Calcula el valor más pequeño. Para la sucesión de columnas, la secuencia de comparación se usa para comparar las sucesiones.
MAX	Calcula el valor más grande. Para la sucesión de columnas, la secuencia de comparación se usa para comparar las sucesiones.

**WHERE vs. HAVING**  
 use la cláusula WHERE para las condiciones que puedan aplicarse a filas individuales. Use la cláusula HAVING para las condiciones que puedan aplicarse únicamente a grupos. Las condiciones en la cláusula HAVING deben incluir funciones agregadas, mientras que las condiciones en la cláusula WHERE no puede incluir funciones agregadas.

no aceptan la palabra clave DISTINCT dentro de funciones agregadas. El capítulo 9 presenta una formulación alternativa en Access SQL para compensar la incapacidad de usar la palabra clave DISTINCT dentro de la función COUNT.

Los ejemplos 4.13 y 4.14 contrastan las cláusulas WHERE y HAVING. En el ejemplo 4.13, la cláusula WHERE selecciona a los estudiantes de mayor grado (básicos o avanzados) antes de agrupar por carrera. Dado que la cláusula WHERE elimina a los estudiantes antes de que suceda la agrupación, sólo se agrupa a los estudiantes de mayor grado. En el ejemplo 4.14, la condición HAVING retiene a los grupos con un promedio mayor que 3.1. La cláusula HAVING aplica para los grupos de filas, mientras que la cláusula WHERE aplica para las filas individuales. Para usar la cláusula HAVING, debe existir la cláusula GROUP BY.

### EJEMPLO 4.13

#### Agrupación con condiciones de fila

Agrupe el promedio de los estudiantes de grados avanzados (*junior* o *senior*) por especialidad.

```
SELECT StdMajor, AVG(StdGPA) AS AvgGpa
  FROM Student
 WHERE StdClass = 'JR' OR StdClass = 'SR'
 GROUP BY StdMajor
```

StdMajor	AvgGPA
ACCT	3.5
FIN	2.800000031789
IS	3.149999976158

### EJEMPLO 4.14

#### Agrupación con condiciones de fila y grupo

Agrupe el promedio de los estudiantes de grados avanzados (*junior* o *senior*) por especialidad. Sólo liste las carreras con promedios mayores a 3.1.

```
SELECT StdMajor, AVG(StdGPA) AS AvgGpa
  FROM Student
 WHERE StdClass IN ('JR', 'SR')
 GROUP BY StdMajor
 HAVING AVG(StdGPA) > 3.1
```

StdMajor	AvgGPA
ACCT	3.5
IS	3.149999976158

#### recordatorio

#### HAVING

la cláusula HAVING debe estar precedida por la cláusula GROUP BY.

Otro punto acerca de los ejemplos 4.13 y 4.14 es el uso del operador OR en comparación con el operador IN (operador de elementos de conjuntos). La condición WHERE del ejemplo 4.13 y 4.14 extrae las mismas filas. La condición IN es verdadera si *StdClass* coincide con cualquier valor de la lista que está dentro del paréntesis. El capítulo 9 proporciona una explicación adicional acerca del operador IN para las consultas anidadas.

Para agrupar todas las filas, las funciones agregadas se pueden utilizar en una sentencia SELECT sin una cláusula GROUP BY, como se demuestra en el ejemplo 4.15. El resultado siempre es una sola fila que contiene únicamente los cálculos agregados.

### EJEMPLO 4.15

#### Agrupación de todas las filas

Liste el número de estudiantes de grados avanzados y sus promedios.

```
SELECT COUNT(*) AS StdCnt, AVG(StdGPA) AS AvgGPA
  FROM Student
 WHERE StdClass = 'JR' OR StdClass = 'SR'
```

StdCnt	AvgGPA
8	3.0625

Algunas veces es útil agrupar más de una columna como se demuestra en el ejemplo 4.16. El resultado muestra una fila para cada combinación de *StdMajor* y *StdClass*. Algunas filas tienen el mismo valor para ambos cálculos agregados, ya que sólo hay un renglón asociado en la tabla *Student*. Por ejemplo, sólo existe un renglón para la combinación de ('ACCT', 'JR').

### EJEMPLO 4.16

#### Agrupación en dos columnas

Agrupe el promedio mínimo y máximo de los estudiantes por carrera y grupo.

```
SELECT StdMajor, StdClass, MIN(StdGPA) AS MinGPA, MAX(StdGPA) AS
      MaxGPA
  FROM Student
 GROUP BY StdMajor, StdClass
```

StdMajor	StdClass	MinGPA	MaxGPA
ACCT	JR	3.5	3.5
ACCT	SO	3.3	3.3
FIN	JR	2.5	2.7
FIN	SR	3.2	3.2
IS	FR	3	3
IS	JR	3.6	3.6
IS	SO	3.8	3.8
IS	SR	2.2	4

Una poderosa combinación es utilizar agrupación con enlaces. No existe alguna razón para restringir el agrupamiento a sólo una tabla. Muchas veces la información más útil se obtiene de la agrupación de las filas resultantes de un enlace. El ejemplo 4.17 muestra la agrupación aplicada a un enlace entre *Course* y *Offering*. Es importante remarcar que el enlace se ejecuta antes de que ocurra la agrupación. Por ejemplo, después del enlace existen seis registros para BUSINESS PROGRAMMING. Como las consultas combinadas con los enlaces y la agrupación pueden ser difíciles de comprender, la sección 4.3 proporciona una explicación con mayor detalle.

**EJEMPLO 4.17  
(Access)****Combinación de agrupación y enlaces**

Agrupe el número de cursos de IS ofrecidos según la descripción del curso.

```
SELECT CrsDesc, COUNT(*) AS OfferCount
  FROM Course, Offering
 WHERE Course.CourseNo = Offering.CourseNo
   AND Course.CourseNo LIKE 'IS*'
 GROUP BY CrsDesc
```

CrsDesc	OfferCount
FUNDAMENTOS DE PROGRAMACIÓN DE NEGOCIOS	6
FUNDAMENTOS DE ADMINISTRACIÓN DE BASES DE DATOS	2
ANÁLISIS DE SISTEMAS	2

**EJEMPLO 4.17  
(Oracle)****Combinación de agrupación y enlaces**

Agrupe el número de cursos de IS ofrecidos según la descripción del curso.

```
SELECT CrsDesc, COUNT(*) AS OfferCount
  FROM Course, Offering
 WHERE Course.CourseNo = Offering.CourseNo
   AND Course.CourseNo LIKE 'IS%'
 GROUP BY CrsDesc
```

**4.2.4 Mejorar el formato de los resultados**

Terminamos esta sección con dos partes de la sentencia SELECT que pueden mejorar el formato de los resultados. Los ejemplos 4.18 y 4.19 demuestran la clasificación que usa la cláusula ORDER BY. La secuencia de clasificación (*sort*) depende del tipo de fecha del campo a ordenar

**EJEMPLO 4.18****Clasificación en una sola columna**

Liste el promedio, nombre, ciudad y estado de los estudiantes de nivel básico. Ordene el resultado por promedio de forma ascendente.

```
SELECT StdGPA, StdFirstName, StdLastName, StdCity, StdState
  FROM Student
 WHERE StdClass = 'JR'
 ORDER BY StdGPA
```

StdGPA	StdFirstName	StdLastName	StdCity	StdState
2.50	ROBERTO	MORALES	SEATTLE	WA
2.70	BOB	NORBERT	BOTHELL	WA
3.50	CANDY	KENDALL	TACOMA	WA
3.60	MARIAH	DODGE	SEATTLE	WA

(numérico para los tipos de datos numéricos, secuencias ASCII para los campos de cadenas de caracteres, y secuencias de tipo calendario para los campos de datos). Por omisión, la clasificación se realiza de forma ascendente. La palabra clave DESC se puede usar después del nombre de una columna para que su orden sea descendiente, tal como se muestra en el ejemplo 4.19.

### EJEMPLO 4.19

#### Clasificación de dos columnas de forma descendente

Liste el rango, salario, nombre y departamento del profesor. Ordene el resultado de forma ascendente (alfabéticamente) según el rango y de forma descendente por el salario.

```
SELECT FacRank, FacSalary, FacFirstName, FacLastName, FacDept
      FROM Faculty
 ORDER BY FacRank, FacSalary DESC
```

FacRank	FacSalary	FacFirstName	FacLastName	FacDept
ASSC	75000.00	JULIA	MILLS	FIN
ASSC	70000.00	LEONARD	FIBON	MS
ASST	40000.00	CRISTOPHER	COLAN	MS
ASST	35000.00	LEONARD	VINCE	MS
PROF	120000.00	VICTORIA	EMMANUEL	MS
PROF	65000.00	NICKI	MACON	FIN

### ORDER BY vs. DISTINCT

use la cláusula ORDER BY para ordenar el resultado de una tabla en una o más columnas. Utilice la palabra clave DISTINCT para eliminar los resultados duplicados.

Algunos estudiantes confunden ORDER BY y GROUP BY. En la mayoría de los sistemas GROUP BY tiene efectos colaterales al ordenar por las columnas agrupadas. No se debe depender de este efecto colateral. Si sólo se requiere ordenar, utilice ORDER BY en lugar de GROUP BY. Si quiere ordenar y agrupar, utilice tanto GROUP BY como ORDER BY.

Otra manera de mejorar el formato del resultado es eliminar las filas duplicadas. Por definición, SQL no elimina las filas duplicadas. Se pueden evitar las filas duplicadas incluyendo las llaves primarias de las tablas resultantes. Existe un sinnúmero de situaciones en donde la llave primaria no aparece en el resultado. El ejemplo 4.21 demuestra el uso de la palabra clave DISTINCT para eliminar los duplicados que aparecen en el resultado del ejemplo 4.20.

### EJEMPLO 4.20

#### Resultado con duplicados

Liste la ciudad y el estado de los profesores.

```
SELECT FacCity, FacState
      FROM Faculty
```

FacCity	FacState
SEATTLE	WA
BOTHELL	WA
SEATTLE	WA
BELLEVUE	WA
SEATTLE	WA
SEATTLE	WA

**EJEMPLO 4.21****Eliminación de duplicados con DISTINCT**

Liste la combinación única de ciudad y estado de la tabla *Faculty*.

```
SELECT DISTINCT FacCity, FacState
  FROM Faculty
```

FacCity	FacState
BELLEVUE	WA
BOTHELL	WA
SEATTLE	WA

## 4.3 Proceso de evaluación conceptual para las sentencias SELECT

**proceso de evaluación conceptual**  
 la secuencia de las operaciones y de las tablas intermedias utilizadas para deducir los resultados de una sentencia SELECT. El proceso de evaluación conceptual puede ayudarle a obtener un conocimiento básico de la sentencia SELECT, así como a comprender problemas más difíciles.

Para desarrollar una comprensión más clara de la sentencia SELECT, es útil comprender el proceso de evaluación conceptual o la secuencia de pasos a seguir para generar el resultado deseado. El proceso de evaluación conceptual describe las operaciones (la mayoría son operaciones de álgebra relacional) que generan tablas intermedias que conducen a la tabla resultante. Puede serle de utilidad hacer referencia al proceso de evaluación conceptual cuando haya aprendido a generar las sentencias SELECT. Después de que obtenga las habilidades iniciales con la sentencia SELECT, no necesitará hacer referencia al proceso de evaluación conceptual, excepto para obtener un bosquejo acerca de problemas difíciles.

Para demostrar el proceso de evaluación conceptual, considere el ejemplo 4.22, el cual incluye muchas partes de la sentencia SELECT. Este ejemplo involucra muchas tablas (*Enrollment* y *Offering* en la cláusula FROM), condiciones de filas (después de WHERE), funciones agragadas (COUNT y AVG) a grupos de filas (GROUP BY), una condición de agrupación (después de HAVING) y la clasificación del resultado final (ORDER BY).

**EJEMPLO 4.22  
(Access)****Ilustre múltiples partes de la sentencia SELECT**

Liste el número de curso, número de oferta y el promedio del grado de los estudiantes inscritos en los cursos de IS ofrecidos durante el otoño de 2005 con por lo menos un estudiante inscrito. Ordene el resultado por número de curso en orden ascendente y el promedio del grado en orden descendente. La versión de Oracle del ejemplo 4.22 es idéntica excepto por % en lugar de \* que funciona como carácter comodín.

```
SELECT CourseNo, Offering.OfferNo, AVG(EnrGrade) AS AvgGrade
  FROM Enrollment, Offering
 WHERE CourseNo LIKE 'IS*' AND OffYear = 2005
       AND OffTerm = 'FALL'
       AND Enrollment.OfferNo = Offering.OfferNo
 GROUP BY CourseNo, Offering.OfferNo
 HAVING COUNT(*) > 1
 ORDER BY CourseNo, 3 DESC
```

En la cláusula ORDER BY, observe que el número 3 es la segunda columna a ordenar. El número 3 significa que se ordena por la tercera columna (*AvgGrade*) en la sentencia SELECT. Algunos DBMS no permiten expresiones agregadas o alias (*AvgGrade*) en la cláusula ORDER BY.

**TABLA 4.10**  
**Ejemplo de la tabla**  
*Offering*

OfferNo	CourseNo	OffYear	OffTerm
1111	IS480	2005	FALL
2222	IS480	2005	FALL
3333	IS320	2005	FALL
5555	IS480	2006	WINTER
6666	IS320	2006	SPRING

**TABLA 4.11**  
**Ejemplo de la tabla**  
*Enrollment*

StdSSN	OfferNo	EnrGrade
111-11-1111	1111	3.1
111-11-1111	2222	3.5
111-11-1111	3333	3.3
111-11-1111	5555	3.8
222-22-2222	1111	3.2
222-22-2222	2222	3.3
333-33-3333	1111	3.6

**TABLA 4.12**  
**Ejemplo del resultado**  
4.22

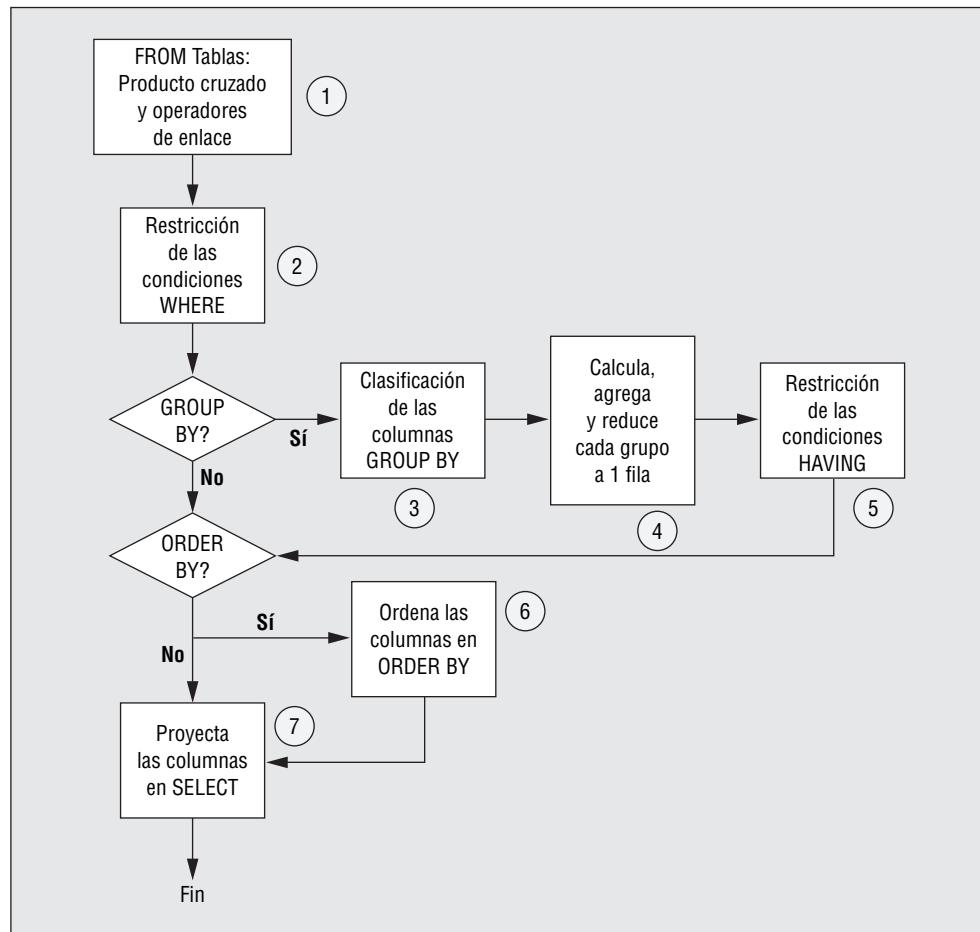
CourseNo	OfferNo	AvgGrade
IS480	2222	3.4
IS480	1111	3.3

Desde la tabla 4.10 hasta la 4.12 se muestran las tablas iniciales y sus resultados. Para que usted pueda comprender de forma más clara el proceso para derivar los resultados sólo se han utilizado los insumos pequeños y los resultados de las tablas. No se necesitan tablas extensas para ilustrar el proceso de evaluación conceptual.

El proceso de evaluación conceptual es una secuencia de operaciones tal como se indica en la figura 4.2. Este proceso es más conceptual que real, pues la mayoría de los compiladores de SQL pueden generar la misma salida usando muchos atajos. No revisaremos los atajos específicos debido a que no son matemáticos sino específicos para cada sistema, así como orientados al desempeño. El proceso de evaluación conceptual provee las bases para la comprensión del significado de las sentencias SQL que son independientes del sistema y de los elementos de desempeño. El resto de esta sección aplica el proceso de evaluación conceptual al ejemplo 4.22.

1. El primer paso en el proceso conceptual combina las tablas de la cláusula FROM con el producto cruzado y los operadores de enlace. En el ejemplo 4.22, es necesaria la operación de productos cruz debido a que se listan dos tablas. No es necesaria una operación de enlace ya que la palabra clave INNER JOIN no aparece en la sentencia FROM. Recuerde que el operador del producto cruz muestra todas las filas posibles que se encuentran al combinar dos tablas. La tabla resultante contiene el producto de los números de fila y la suma de las columnas. En este caso, los productos cruz tienen 35 renglones ( $5 \times 7$ ) y 7 columnas ( $3 + 4$ ). La tabla 4.13 muestra un resultado parcial. A manera de ejercicio se le sugiere que obtenga el resultado completo. Como abreviación de la notación aquí el nombre de la tabla (abreviado como *E* y *O*) se antepone al nombre de la columna para *OfferNo*.
2. El segundo paso usa el operador de restricción para extraer las filas que satisfacen las condiciones de la cláusula WHERE del resultado del paso 1. Existen cuatro condiciones: una condición de enlace en *OfferNo*, una condición en *CourseNo*, una condición en *OffYear*, y una condición en *OffTerm*. Observe que la condición en *CourseNo* incluye el

**FIGURA 4.2**  
Diagrama de flujo del proceso de evaluación conceptual



**TABLA 4.13**  
Resultado parcial  
del paso 1 para las  
primeras dos filas de  
*Offering* (1111 y 2222)

O.OfferNo	CourseNo	OffYear	OffTerm	StdSSN	E.OfferNo	EnrGrade
1111	IS480	2005	FALL	111-11-1111	1111	3.1
1111	IS480	2005	FALL	111-11-1111	2222	3.5
1111	IS480	2005	FALL	111-11-1111	3333	3.3
1111	IS480	2005	FALL	111-11-1111	5555	3.8
1111	IS480	2005	FALL	222-22-2222	1111	3.2
1111	IS480	2005	FALL	222-22-2222	2222	3.3
1111	IS480	2005	FALL	333-33-3333	1111	3.6
2222	IS480	2005	FALL	111-11-1111	1111	3.1
2222	IS480	2005	FALL	111-11-1111	2222	3.5
2222	IS480	2005	FALL	111-11-1111	3333	3.3
2222	IS480	2005	FALL	111-11-1111	5555	3.8
2222	IS480	2005	FALL	222-22-2222	1111	3.2
2222	IS480	2005	FALL	222-22-2222	2222	3.3
2222	IS480	2005	FALL	333-33-3333	1111	3.6

carácter comodín (\*). Cualquier número de curso que comience con IS coincide con esta condición. La tabla 4.14 muestra que el resultado del producto cruzado (35 filas) se reduce a seis filas.

3. El tercer paso ordena el resultado del paso 2 por las columnas especificadas en la cláusula GROUP BY. La cláusula GROUP BY indica que la salida se debe relacionar con el grupo

**TABLA 4.14**  
**Resultado del paso 2**

O.OfferNo	CourseNo	OffYear	OffTerm	StdSSN	E.OfferNo	EnrGrade
1111	IS480	2005	FALL	111-11-1111	1111	3.1
2222	IS480	2005	FALL	111-11-1111	2222	3.5
1111	IS480	2005	FALL	222-22-2222	1111	3.2
2222	IS480	2005	FALL	222-22-2222	2222	3.3
1111	IS480	2005	FALL	333-33-3333	1111	3.6
3333	IS320	2005	FALL	111-11-1111	3333	3.3

**TABLA 4.15**  
**Resultado del paso 3**

CourseNo	O.OfferNo	OffYear	OffTerm	StdSSN	E.OfferNo	EnrGrade
IS320	3333	2005	FALL	111-11-1111	3333	3.3
IS480	1111	2005	FALL	111-11-1111	1111	3.1
IS480	1111	2005	FALL	222-22-2222	1111	3.2
IS480	1111	2005	FALL	333-33-3333	1111	3.6
IS480	2222	2005	FALL	111-11-1111	2222	3.5
IS480	2222	2005	FALL	222-22-2222	2222	3.3

de filas en lugar de renglones individuales. Si la salida se relaciona con registros individuales en lugar de grupos de registros, se omite la cláusula GROUP BY. Cuando se utiliza la cláusula GROUP BY debe incluir *cada* columna de la cláusula SELECT, excepto las expresiones que involucran a una función agregada.<sup>4</sup> La tabla 4.15 muestra el resultado del paso 2 ordenado por *CourseNo* y *O.OfferNo*. Observe que las columnas han sido reacomodadas para facilitar la lectura del resultado.

4. El cuarto paso solamente es necesario si existe una cláusula GROUP BY. El cuarto paso calcula la(s) función(es) agregada(s) para cada grupo de filas y reduce cada grupo a una sola fila. Todos los registros de un grupo tienen los mismos valores para las columnas GROUP BY. En la tabla 4.16 existen tres grupos {<IS320,3333>, <IS480, 1111>, <IS480,2222>}. Las columnas calculadas se suman para las funciones agregadas de las cláusulas SELECT y HAVING. La tabla 4.16 muestra dos columnas nuevas para la función AVG de la cláusula SELECT, así como para la función COUNT de la cláusula HAVING. Observe que las columnas restantes se eliminan en este punto, ya que no se necesitan en los siguientes pasos.
5. El quinto paso elimina las filas que no satisfacen la condición de la cláusula HAVING. La tabla 4.17 muestra que el primer registro de la tabla 4.16 se elimina porque no cumple con la condición de la cláusula HAVING. Observe que la cláusula HAVING especifica las operaciones de restricción para grupos o filas. La cláusula HAVING no puede presentarse sin que la preceda antes la cláusula GROUP BY. Las condiciones de la cláusula HAVING siempre se relacionan con grupos de filas y no con filas individuales. Comúnmente las condiciones de la cláusula HAVING incluyen funciones agregadas.
6. El sexto paso ordena los resultados de acuerdo con la cláusula ORDER BY. Observe que la cláusula ORDER BY es opcional. La tabla 4.18 muestra el resultado de la tabla después de haberla ordenado.
7. El séptimo paso realiza el proceso de una proyección final. Las columnas que aparecen en el resultado del paso 6 se eliminan cuando no aparezcan en la cláusula SELECT. La tabla 4.19 (idéntica a la tabla 4.12) muestra el resultado obtenido después de la proyección del paso 6. La columna *Count(\*)* se elimina ya que no aparece en la sentencia SELECT. El séptimo paso (proyección) ocurre después del sexto paso (ordenación), ya que la cláusula ORDER BY puede incluir las columnas que no aparecen en la lista de la sentencia SELECT.

<sup>4</sup> En otras palabras, cuando se usa la cláusula GROUP BY, cada columna de la cláusula SELECT debe estar en la cláusula GROUP BY o debe ser parte de alguna expresión con una función agregada.

**TABLA 4.16**  
Resultado del paso 4

CourseNo	O.OfferNo	AvgGrade	Count(*)
IS320	3333	3.3	1
IS480	1111	3.3	3
IS480	2222	3.4	2

**TABLA 4.17**  
Resultado del paso 5

CourseNo	O.OfferNo	AvgGrade	Count(*)
IS480	1111	3.3	3
IS480	2222	3.4	2

**TABLA 4.18**  
Resultado del paso 6

CourseNo	O.OfferNo	AvgGrade	Count(*)
IS480	2222	3.4	3
IS480	1111	3.3	2

**TABLA 4.19**  
Resultado del paso 7

CourseNo	O.OfferNo	AvgGrade
IS480	2222	3.4
IS480	1111	3.3

Esta sección finaliza con un comentario acerca de las tres principales lecciones del proceso conceptual de evaluaciones. Es más importante recordar estas lecciones que los detalles específicos del proceso conceptual.

- La conceptuación de GROUP BY ocurre después de la sentencia WHERE. Si se tiene algún error en la sentencia SELECT que incluya la sentencia WHERE o GROUP BY, el problema probablemente esté en la cláusula WHERE. Puede revisar los resultados intermedios después de la cláusula WHERE al ejecutar la sentencia SELECT sin la cláusula GROUP BY.
- La agrupación sólo sucede una vez en el proceso de evaluación. Si su problema incluye más de un cálculo independiente agregado, puede que necesite más de una sentencia SELECT.
- El uso de tablas de los ejemplos puede ayudarle a analizar problemas difíciles. Generalmente no es necesario revisar el proceso de evaluación por completo. En su lugar, use tablas de ejemplo que le ayuden a comprender sólo la parte difícil. La sección 4.5 y el capítulo 9 ilustran el uso de tablas de ejemplo para ayudarle a analizar problemas difíciles.

## 4.4 Preguntas críticas para la generación de consultas

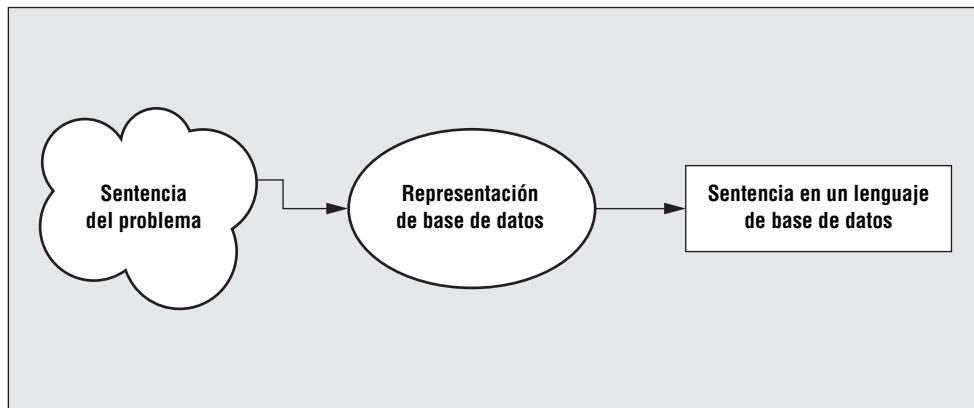
### preguntas críticas para la generación de consultas

proporcionan una lista para convertir la sentencia de algún problema en una representación de base de datos con tablas, columnas, operaciones para conectar tablas y requerimientos de agrupación de filas.

El proceso de evaluación conceptual ilustrado en la figura 4.2 debe ayudarle a entender el significado de la mayoría de las sentencias SELECT, pero probablemente no le ayude a generar consultas. La generación de consultas incluye la conversión de la sentencia de algún problema en una sentencia de lenguaje de base de datos, como SQL, tal como se muestra en la figura 4.3. Entre la sentencia del problema y la sentencia del lenguaje de base de datos, usted transforma la sentencia del problema en una representación de base de datos. Comúnmente, la parte más difícil es convertir la sentencia del problema en una representación de base de datos. Esta conversión incluye un conocimiento detallado de las tablas y sus relaciones, así como de mucha atención a posibles ambigüedades de la sentencia del problema. Las preguntas críticas que se presentan en esta sección proporcionan un proceso estructurado que convierte la sentencia del problema en una representación de base de datos.

En la conversión de la sentencia del problema en una representación de base de datos, usted debe responder algunas preguntas críticas. La tabla 4.20 resume el análisis de las preguntas críticas.

**FIGURA 4.3**  
Proceso de generación de consultas



**TABLA 4.20**  
Resumen de las preguntas críticas para la generación de consultas

Preguntas	Consejos para el análisis
¿Qué tablas se necesitan?	Relacione las columnas con los requerimientos de los datos de salida y con las condiciones a probar. Si las tablas no están relacionadas directamente, identifique las tablas intermedias para proveer una ruta de enlace entre las tablas.
¿Cómo se relacionan las tablas?	La mayoría de las tablas se combinan mediante el uso de una llave primaria de una tabla madre relacionada con una llave externa de una tabla hija. Los problemas más difíciles pueden incluir otras condiciones de enlace, al igual que otros operadores de combinación (enlace externo, diferencia o división).
¿La salida requiere de la relación de filas individuales o de filas agrupadas?	Identifique las funciones agregadas utilizadas en los requerimientos de los datos de salida y en las condiciones a probar. La sentencia SELECT requiere una cláusula GROUP BY si se necesitan funciones agregadas. La cláusula HAVING es necesaria si las condiciones usan las funciones agregadas.

¿Qué tablas se necesitan? Para la primera pregunta debe relacionar los requerimientos de datos en columnas y tablas. Debe identificar las columnas que se necesitan para la salida y las tablas intermedias necesarias para conectar otras tablas. Por ejemplo, si usted quiere enlazar las tablas *Student* y *Offering*, debe incluir la tabla *Enrollment*, ya que proporciona la conexión entre estas tablas. Las tablas *Student* y *Offering* no se pueden enlazar de forma directa. Todas las tablas necesarias para la consulta deben enlistarse en la cláusula FROM.

¿Cómo se relacionan las tablas? Segunda pregunta: la mayoría de las tablas se combinan por un operador de enlace. En el capítulo 9 utilizará los operadores del enlace externo (*outer join*), diferencia y división para combinar las tablas. Por ahora, sólo enfóquese en la combinación de las tablas con enlaces. Requerirá identificar las columnas relacionadas para cada enlace. En la mayoría de los enlaces, la llave primaria de la tabla madre se relaciona con una llave externa de una tabla hija relacionada. En algunas situaciones, la llave primaria de la tabla madre contiene muchas columnas. En este caso necesita relacionar ambas columnas. En ocasiones, las columnas relacionadas no involucran una combinación de llave primaria con llave foránea. Puede realizar un enlace mientras las columnas relacionadas tengan tipos de datos compatibles. Por ejemplo, cuando se enlazan las tablas de clientes de diferentes bases de datos es posible que no tengan una llave primaria común. Quizá sea necesario el enlace de otros campos, tales como nombre, dirección, etcétera.

¿La salida se relaciona con filas individuales o con grupos de filas? Para la tercera pregunta, revise los cálculos que involucren funciones agregadas en la sentencia del problema. Por ejemplo, el problema “liste los nombres y los promedios de los grados de los estudiantes” contiene un cálculo agregado. Los problemas que se refieren a una función agregada indican que la salida se relaciona con un grupo de filas. Por ende, la sentencia SELECT requiere una cláusula GROUP BY. Si el problema contiene condiciones con funciones agregadas, una cláusula HAVING debe

acompañar a la cláusula GROUP BY. Por ejemplo, el problema “liste el número de cursos ofrecidos, impartidos con más de 30 estudiantes”, necesita una cláusula HAVING con una condición que incluya la función COUNT.

Después de responder estas preguntas, estará listo para convertir la representación de la base de datos en una sentencia del lenguaje de base de datos. Para ayudarle en este proceso, deberá desarrollar un conjunto de sentencias para cada tipo de operador del álgebra relacional, usando una base de datos que entienda bien. Por ejemplo, debería tener sentencias para problemas que incluyan los operadores de enlace, enlaces con agrupaciones y enlaces con condiciones de agrupación. Conforme aumente su conocimiento de SQL, la conversión será más fácil para la mayoría de los problemas. Para los problemas difíciles, como los ya comentados en la sección 4.5 y en el capítulo 9, puede que sea necesaria la revisión de problemas similares, ya que no son comunes los problemas difíciles.

## 4.5 Mejorando las habilidades de generación de consultas mediante ejemplos

---

Apliquemos las habilidades y conocimientos en la generación de consultas de la sentencia SELECT a problemas más difíciles. Todos los problemas de esta sección incluyen las partes de la sentencia SELECT que ya analizamos en las secciones 4.2 y 4.3. Los problemas incluyen aspectos más difíciles, tales como los enlaces entre más de dos tablas, agrupaciones después de varios enlaces entre tablas, enlaces de tablas con ellas mismas y operadores tradicionales de conjuntos.

### **estilo del producto cruz**

liste las tablas de la cláusula FROM y las condiciones de enlace en la cláusula WHERE. El estilo del producto cruz es fácil de leer, pero no tiene el respaldo para las operaciones de enlace externo.

### 4.5.1 Enlaces de tablas múltiples con el estilo del producto cruz

Empezamos con un número de problemas de enlaces que se formulan usando los operadores del producto cruz en la cláusula FROM. A esta forma de definir los enlaces se le conoce como estilo del producto cruz por la implementación de los operadores del producto cruz. La siguiente subsección utiliza operadores de enlace en la cláusula FROM para contrastar las formas en que los enlaces pueden expresarse.

#### EJEMPLO 4.23

#### Enlazando dos tablas

Liste el nombre del estudiante, número de curso ofrecido y la calificación del estudiante con una calificación  $\geq 3.5$  en un curso ofrecido.

```
SELECT StdFirstName, StdLastName, OfferNo, EnrGrade
  FROM Student, Enrollment
 WHERE EnrGrade >= 3.5
   AND Student.StdSSN = Enrollment.StdSSN
```

StdFirstName	StdLastName	OfferNo	EnrGrade
CANDY	KENDALL	1234	3.5
MARIAH	DODGE	1234	3.8
HOMER	WELLS	4321	3.5
ROBERTO	MORALES	4321	3.5
BOB	NORBERT	5679	3.7
ROBERTO	MORALES	5679	3.8
MARIAH	DODGE	6666	3.6
LUKE	BRAZZI	7777	3.7
BOB	NORBERT	9876	3.5
WILLIAM	PILGRIM	9876	4

En el ejemplo 4.23 algunas filas de estudiantes aparecen más de una vez en el resultado. Por ejemplo, Roberto Morales aparece dos veces. Debido a la relación de 1-M entre las tablas de *Student* y *Enrollment*, la fila *Student* puede relacionarse con varias filas de *Enrollment*.

Los ejemplos 4.24 y 4.25 ilustran la eliminación de duplicados después del enlace. En el ejemplo 4.24 algunos estudiantes aparecen más de una vez, como en el ejemplo 4.23. Esto se debe a que en la salida sólo se usan las columnas de la tabla *Student*. Las filas duplicadas pueden aparecer en el resultado cuando enlaza una tabla madre con una tabla hija y sólo muestra las columnas de la tabla madre en el resultado. Para eliminar las filas duplicadas puede utilizar la palabra clave DISTINCT, como se muestra en el ejemplo 4.25.

#### EJEMPLO 4.24

#### Enlaces con duplicados

Liste el nombre de los estudiantes que tengan una calificación  $\geq 3.5$  en el curso ofrecido.

```
SELECT StdFirstName, StdLastName
  FROM Student, Enrollment
 WHERE EnrGrade >= 3.5
   AND Student.StdSSN = Enrollment.StdSSN
```

StdFirstName	StdLastName
CANDY	KENDALL
MARIAH	DODGE
HOMER	WELLS
ROBERTO	MORALES
BOB	NORBERT
ROBERTO	MORALES
MARIAH	DODGE
LUKE	BRAZZI
BOB	NORBERT
WILLIAM	PILGRIM

#### EJEMPLO 4.25

#### Enlaces con duplicados eliminados

Liste los nombres de los estudiantes (sin duplicarlos) que tengan una calificación  $\geq 3.5$  en el curso ofrecido.

```
SELECT DISTINCT StdFirstName, StdLastName
  FROM Student, Enrollment
 WHERE EnrGrade >= 3.5
   AND Student.StdSSN = Enrollment.StdSSN
```

StdFirstName	StdLastName
BOB	NORBERT
CANDY	KENDALL
HOMER	WELLS
LUKE	BRAZZI
MARIAH	DODGE
ROBERTO	MORALES
WILLIAM	PILGRIM

Los ejemplos 4.26 a 4.29 ilustran los problemas que incluyen más de dos tablas. En estos problemas es importante identificar las tablas de la cláusula FROM. Debe estar seguro de examinar las condiciones a probar, al igual que las columnas del resultado. En el ejemplo 4.28 se necesita la tabla *Enrollment*, aun cuando no proporcione columnas del resultado o condiciones de prueba.

**EJEMPLO 4.26****Enlaces entre tres tablas con columnas de sólo dos tablas**

Liste el nombre del estudiante y el número del curso ofrecido en el cual la calificación sea mayor a 3.7 y que se haya impartido en el oto de 2005.

```
SELECT StdFirstName, StdLastName, Enrollment.OfferNo
FROM Student, Enrollment, Offering
WHERE Student.StdSSN = Enrollment.StdSSN
    AND Offering.OfferNo = Enrollment.OfferNo
    AND OffYear = 2005 AND OffTerm = 'FALL'
    AND EnrGrade >= 3.7
```

StdFirstName	StdLastName	OfferNo
MARIAH	DODGE	1234

**EJEMPLO 4.27****Enlaces entre tres tablas con columnas de sólo dos tablas**

Liste el horario de enseñanza para Leonard Vince del o yo de 2005. Para cada curso, liste el n umero de cursos ofrecidos, n umero de curso y n umero de unidades, d as, ubicaci on y horario.

```
SELECT OfferNo, Offering.CourseNo, CrsUnits, OffDays, OffLocation, OffTime
FROM Faculty, Course, Offering
WHERE Faculty.FacSSN = Offering.FacSSN
    AND Offering.CourseNo = Course.CourseNo
    AND OffYear = 2005 AND OffTerm = 'FALL'
    AND FacFirstName = 'LEONARD'
    AND FacLastName = 'VINCE'
```

OfferNo	CourseNo	CrsUnits	OffDays	OffLocation	OffTime
1234	IS320	4	MW	BLM302	10:30 AM
4321	IS320	4	TTH	BLM214	3:30 PM

**EJEMPLO 4.28****Enlaces entre cuatro tablas**

Liste el horario de cursos para Bob Norbert de la primavera de 2006. Para cada curso, liste el n umero de curso ofrecido, n umero de curso, d as, ubicaci on, horario y nombre del profesor.

```
SELECT Offering.OfferNo, Offering.CourseNo, OffDays, OffLocation,
    OffTime, FacFirstName, FacLastName
FROM Faculty, Offering, Enrollment, Student
WHERE Offering.OfferNo = Enrollment.OfferNo
    AND Student.StdSSN = Enrollment.StdSSN
    AND Faculty.FacSSN = Offering.FacSSN
    AND OffYear = 2006 AND OffTerm = 'SPRING'
    AND StdFirstName = 'BOB'
    AND StdLastName = 'NORBERT'
```

OfferNo	CourseNo	OffDays	OffLocation	OffTime	FacFirstName	FacLastName
5679	IS480	TTH	BLM412	3:30 PM	CRISTOPHER	COLAN
9876	IS460	TTH	BLM307	1:30 PM	LEONARD	FIBON

**EJEMPLO 4.29****Enlaces entre cinco tablas**

Liste el horario de cursos para Bob Norbert de la primavera de 2006. Para cada curso, liste el número de curso ofrecido, número de curso, días, ubicación, horario y nombre del profesor.

```
SELECT Offering.OfferNo, Offering.CourseNo, OffDays, OffLocation, OffTime,
       CrsUnits, FacFirstName, FacLastName
  FROM Faculty, Offering, Enrollment, Student, Course
 WHERE Faculty.FacSSN = Offering.FacSSN
   AND Offering.OfferNo = Enrollment.OfferNo
   AND Student.StdSSN = Enrollment.StdSSN
   AND Offering.CourseNo = Course.CourseNo
   AND OffYear = 2006 AND OffTerm = 'SPRING'
   AND StdFirstName = 'BOB'
   AND StdLastName = 'NORBERT'
```

OfferNo	CourseNo	OffDays	OffLocation	OffTime	CrsUnits	FacFirstName	FacLastName
5679	IS480	TTH	BLM412	3:30 PM	4	CRISTOPHER	COLAN
9876	IS460	TTH	BLM307	1:30 PM	4	LEONARD	FIBON

La tabla *Enrollment* es necesaria para realizar la conexión de la tabla *Student* con la tabla *Offering*. El ejemplo 4.29 amplía el ejemplo 4.28 agregando los detalles de la tabla *Course*. Las cinco tablas son necesarias para generar las salidas, para probar las condiciones o para conectar otras tablas.

En el ejemplo 4.30 se presenta otra manera de combinar la tabla *Student* y *Faculty*. En el ejemplo 4.28 se demostró que era necesario combinar las tablas *Student*, *Enrollment*, *Offering* y *Faculty* para encontrar al profesor específico para un estudiante. Para encontrar a estudiantes en el cuerpo de profesores (quizá como asistentes de los maestros), se pueden enlazar las tablas

**EJEMPLO 4.30****Enlace de dos tablas sin coincidencia entre una llave primaria y una foránea**

Liste a los estudiantes en el cuerpo de profesores. Incluya todas las columnas de los estudiantes en el resultado.

```
SELECT Student.*
  FROM Student, Faculty
 WHERE StdSSN = FacSSN
```

StdSSN	StdFirstName	StdLastName	StdCity	StdState	StdMajor	StdClass	StdGPA	StdZip
876-54-3210	CRISTOPHER	COLAN	SEATTLE	WA	IS	SR	4.00	98114-1332

**estilo del operador de enlace (*join*)**  
 liste las operaciones de enlace de la cláusula FROM utilizando las palabras clave INNER JOIN y ON. El estilo del operador de enlace puede ser de alguna manera difícil de leer para muchas de las operaciones de enlace, pero soporta operaciones de enlace externas (*outer join*) como se muestra en el capítulo 9.

de forma directa. De esta forma, la combinación de las tablas *Student* y *Faculty* es similar a la operación de intersección. Sin embargo, la intersección no se puede llevar a cabo aquí ya que las tablas *Student* y *Faculty* no son compatibles con el operador unión.

Un detalle acerca del ejemplo 4.30 es el uso del \* después de la palabra clave SELECT. Anteponer un \* con el nombre de alguna tabla y un punto indica que todas las columnas de la tabla especificada deben aparecer en el resultado. El uso de \* sin el prefijo del nombre de una tabla indica que en el resultado están todas las columnas de todas las tablas de la cláusula FROM.

#### 4.5.2 Enlazando tablas múltiples con el estilo del operador de enlace

Como se demostró en la sección 4.2, las operaciones de enlace se pueden expresar directamente en la cláusula FROM utilizando la palabra clave INNER JOIN y ON. El estilo del operador de enlace se puede utilizar para combinar cualquier número de tablas. Para asegurar su comodi-

dad en el uso de este estilo, esta subsección presenta ejemplos de enlaces de tablas múltiples, iniciando con el enlace de las dos tablas del ejemplo 4.31. Observe que estos ejemplos no se ejecutan en versiones de Oracle anteriores a la 9i.

**EJEMPLO 4.31  
(Versiones de Access y Oracle 9i y superiores)**
**Enlace de dos tablas utilizando el estilo del operador de enlace**

Extraiga el nombre, la ciudad y la calificación de los estudiantes con altas calificaciones (mayor o igual a 3.5) en algún curso ofrecido.

```
SELECT StdFirstName, StdLastName, StdCity, EnrGrade
  FROM Student INNER JOIN Enrollment
    ON Student.StdSSN = Enrollment.StdSSN
   WHERE EnrGrade >= 3.5
```

StdFirstName	StdLastName	StdCity	EnrGrade
CANDY	KENDALL	TACOMA	3.5
MARIAH	DODGE	SEATTLE	3.8
HOMER	WELLS	SEATTLE	3.5
ROBERTO	MORALES	SEATTLE	3.5
BOB	NORBERT	BOTHELL	3.7
ROBERTO	MORALES	SEATTLE	3.8
MARIAH	DODGE	SEATTLE	3.6
LUKE	BRAZZI	SEATTLE	3.7
BOB	NORBERT	BOTHELL	3.5
WILLIAM	PILGRIM	BOTHELL	4

El estilo del operador de enlace puede extenderse para cualquier número de tablas. Piense en el estilo del operador de enlace como si escribiera una fórmula complicada con muchos paréntesis. Para añadir otra parte a la fórmula, necesita agregar los argumentos, el operador y otro nivel de paréntesis. Por ejemplo, con la fórmula  $(X + Y)*Z$ , puede añadir otra operación como  $((X + Y)*Z)/W$ . Este mismo principio se puede aplicar con el estilo del operador de enlace. En los ejemplos 4.32 y 4.33 se amplía el ejemplo 4.31 con condiciones adicionales que requieren de otras tablas. En ambos ejemplos se agrega INNER JOIN al final de la operación INNER JOIN anterior. El INNER JOIN también podría haberse agregado al inicio o a la mitad, si así se desea. El orden de las operaciones con INNER JOIN no es importante.

**EJEMPLO 4.32  
(Versiones de Access y Oracle 9i y superiores)**
**Enlace de tres tablas utilizando el estilo del operador de enlace**

Extraiga el nombre, la ciudad y la calificación de los estudiantes con altas calificaciones (mayor o igual a 3.5) en algún curso ofrecido durante el otoño de 2005.

```
SELECT StdFirstName, StdLastName, StdCity, EnrGrade
  FROM ( Student INNER JOIN Enrollment
    ON Student.StdSSN = Enrollment.StdSSN )
    INNER JOIN Offering
      ON Offering.OfferNo = Enrollment.OfferNo
     WHERE EnrGrade >= 3.5 AND OffTerm = 'FALL'
       AND OffYear = 2005
```

StdFirstName	StdLastName	StdCity	EnrGrade
CANDY	KENDALL	TACOMA	3.5
MARIAH	DODGE	SEATTLE	3.8
HOMER	WELLS	SEATTLE	3.5
ROBERTO	MORALES	SEATTLE	3.5

**EJEMPLO 4.33**  
**(Versiones de Access y Oracle 9i y superiores)**

**Enlace cuatro tablas utilizando el estilo del operador de enlace**

Extraiga el nombre, la ciudad y la calificación de los estudiantes con altas calificaciones (mayor o igual a 3.5) en algún curso ofrecido durante el otoño de 2005 e impartido por Leonard Vince.

```
SELECT StdFirstName, StdLastName, StdCity, EnrGrade
  FROM ( (Student INNER JOIN Enrollment
           ON Student.StdSSN = Enrollment.StdSSN )
         INNER JOIN Offering
           ON Offering.OfferNo = Enrollment.OfferNo )
        INNER JOIN Faculty ON Faculty.FacSSN = Offering.FacSSN
 WHERE EnrGrade >= 3.5 AND OffTerm = 'FALL'
       AND OffYear = 2005 AND FacFirstName = 'LEONARD'
       AND FacLastName = 'VINCE'
```

StdFirstName	StdLastName	StdCity	EnrGrade
CANDY	KENDALL	TACOMA	3.5
MARIAH	DODGE	SEATTLE	3.8
HOMER	WELLS	SEATTLE	3.5
ROBERTO	MORALES	SEATTLE	3.5

Los estilos del producto cruz y del operador de enlace se pueden combinar como se demuestra en el ejemplo 4.34. En la mayoría de los casos, es preferible utilizar un estilo u otro, según se requiera.

**EJEMPLO 4.34**  
**(Versiones de Access y Oracle 9i y superiores)**

**Combinación de los estilos del producto cruz y del operador de enlace**

Extraiga el nombre, la ciudad y la calificación de los estudiantes con altas calificaciones (mayor o igual a 3.5) en algún curso ofrecido durante el otoño del 2005 e impartido por Leonard Vince (el resultado será el mismo que en el ejemplo 4.33).

```
SELECT StdFirstName, StdLastName, StdCity, EnrGrade
  FROM ( (Student INNER JOIN Enrollment
           ON Student.StdSSN = Enrollment.StdSSN )
         INNER JOIN Offering
           ON Offering.OfferNo = Enrollment.OfferNo ),
        Faculty
 WHERE EnrGrade >= 3.5 AND OffTerm = 'FALL'
       AND OffYear = 2005 AND FacFirstName = 'LEONARD'
       AND FacLastName = 'VINCE'
       AND Faculty.FacSSN = Offering.FacSSN
```

La elección entre los estilos del producto cruz y del operador de enlace es cuestión de preferencia. En el estilo del producto cruz es fácil ver las tablas en la sentencia de SQL. Para enlaces múltiples, el estilo del operador de enlace puede ser difícil de leer por los paréntesis anidados. La principal ventaja del estilo del operador de enlace es que se pueden generar las consultas incluyendo los enlaces externos, como se describe en el capítulo 9.

Debe sentirse seguro leyendo ambos estilos de enlace aunque escriba las sentencias de SQL utilizando un solo estilo. Tal vez requiera dar mantenimiento a sentencias escritas con ambos estilos. Además, algunos lenguajes visuales de consultas generan el código con uno de los estilos. Por ejemplo, el diseñador de consultas, en el lenguaje de consultas visuales de Microsoft Access, genera el código en el estilo del operador de enlace.

### 4.5.3 Autoenlaces (self-joins) y enlaces múltiples entre dos tablas

#### **autoenlace (self-join)**

un enlace entre una tabla y ella misma (dos copias de la misma tabla). Los autoenlaces son útiles para encontrar las relaciones entre dos filas de la misma tabla.

En el ejemplo 4.35 se demuestra que un *autoenlace* es un enlace incluyendo una tabla consigo misma. Un autoenlace se requiere para encontrar las relaciones entre las filas de la misma tabla. La llave foránea *FacSupervisor* muestra la relación entre las filas de la tabla *Faculty*. Para encontrar el nombre del supervisor de un facultativo, relacione la columna *FacSupervisor* con la columna *FacSSN*. El truco es imaginar que está trabajando con dos copias de la tabla *Faculty*. Una copia juega el rol del subordinado, mientras que la otra juega el rol del superior. En SQL un autoenlace requiere de alias para las tablas (*Subr* y *Supr*) en la cláusula FROM para distinguir entre los dos roles o copias.

#### EJEMPLO 4.35

#### Autoenlace

Liste los profesores que tienen un salario superior al de su supervisor. Liste el número de seguridad social, el nombre y el salario del catedrático y del supervisor.

```
SELECT Subr.FacSSN, Subr.FacLastName, Subr.FacSalary, Supr.FacSSN,
       Supr.FacLastName, Supr.FacSalary
  FROM Faculty Subr, Faculty Supr
 WHERE Subr.FacSupervisor = Supr.FacSSN
   AND Subr.FacSalary > Supr.FacSalary
```

Subr.FacSSN	Subr.FacLastName	Subr.FacSalary	Supr.FacSSN	Supr.FacLastName	Supr.FacSalary
987-65-4321	MILLS	75000.00	765-43-2109	MACON	65000.00

Los problemas que incluyen autoenlaces pueden ser difíciles de comprender. Si tiene problemas para comprender el ejemplo 4.35, utilice el proceso de evaluación conceptual. Empiece con una tabla pequeña de *Faculty*. Copie esta tabla y use los alias *Subr* y *Supr* para distinguir entre las dos copias. Enlace ambas tablas con *Subr.FacSupervisor* y *Supr.FacSSN*. Si lo requiere deduzca el enlace utilizando el operador de producto cruz. Deberá poder observar que cada fila del resultado del enlace muestra a la pareja subordinado y supervisor.

Los problemas que incluyen relaciones autoreferenciadas (únicas) son parte de las consultas ramificadas. En las consultas ramificadas se puede visualizar una tabla como una estructura similar a un árbol o jerárquica. Por ejemplo, la tabla *Faculty* tiene una estructura que muestra una organización jerárquica. En la punta superior se encuentra el rector del colegio. Hasta abajo se encuentran los miembros del profesorado sin subordinados. Estructuras similares se aplican al esquema de cuentas en los sistemas contables, partes de las estructuras de los sistemas de manufactura y redes de rutas de los sistemas de transporte.

Un problema más difícil que un autoenlace es encontrar a todos los subordinados (directa o indirectamente) en la jerarquía de una organización. Este problema se puede resolver en SQL si se conoce el número de niveles de los subordinados. Es necesario un enlace por cada nivel de subordinados. Este problema no se puede resolver en SQL-92 sin conocer el número de niveles de los subordinados, pero en cambio sí se puede solucionar en SQL:2003 utilizando la cláusula WITH RECURSIVE, así como en extensiones propietarias de SQL. En SQL-92, las consultas ramificadas se pueden resolver utilizando SQL dentro de un lenguaje de programación.

El ejemplo 4.36 muestra otro difícil problema de enlaces. Este problema incluye dos enlaces entre las dos mismas tablas (*Offering* y *Faculty*). Se necesitan los alias de los nombres de tablas (*O1* y *O2*) para distinguir entre las dos copias de la tabla *Offering* utilizada en la sentencia.

#### EJEMPLO 4.36

#### Más de un enlace entre tablas que utilizan alias para el nombre de tabla

Liste los nombres de los profesores y el número de curso para el cual el profesor enseña el mismo número de curso que su supervisor durante 2006.

```
SELECT FacFirstName, FacLastName, O1.CourseNo
  FROM Faculty, Offering O1, Offering O2
 WHERE Faculty.FacSSN = O1.FacSSN
   AND Faculty.FacSupervisor = O2.FacSSN
   AND O1.OffYear = 2006 AND O2.OffYear = 2006
   AND O1.CourseNo = O2.CourseNo
```

FacFirstName	FacLastName	CourseNo
LEONARD	VINCE	IS320
LEONARD	FIBON	IS320

Si este problema es demasiado difícil utilice el proceso de evaluación conceptual (figura 4.2) con tablas de ejemplo para obtener un bosquejo. Realice un enlace entre las tablas de ejemplo *Faculty* y *Offering*, después enlace el resultado con otra copia de *Offering* (*O2*) en que se relacione *FacSupervisor* con *O2.FacSSN*. En la tabla resultante, seleccione las filas que relacionen el número de cursos y el año igual a 2006.

#### 4.5.4 Combinando enlaces y agrupación

El ejemplo 4.37 demuestra por qué a veces es necesario agrupar varias columnas. Después de estudiar el ejemplo 4.37, puede estar confundido acerca de la necesidad de agrupar tanto en *OfferNo* como en *CourseNo*. Una explicación simple es que cualquier columna que aparece en la sentencia SELECT debe ser columna agrupada o en expresión agregada. Sin embargo, esta explicación no cuenta la historia completa. La agrupación única de *OfferNo* genera los mismos valores para la columna calculada (*NumStudents*) ya que *OfferNo* es la llave primaria. La agregación de columnas que no son únicas, tales como *CourseNo*, agrega información a cada fila resultante pero no cambia los cálculos agregados. Si no se alcanza a entender este punto, utilice tablas de ejemplo para demostrarlo. Cuando evalúe sus tablas de ejemplo recuerde que el enlace ocurre antes de la agrupación, tal como se indica en el proceso conceptual de evaluación.

**EJEMPLO 4.37****Enlace con agrupaciones en columnas múltiples**

Liste el número de curso, el número de oferta y el número de estudiantes inscritos. Sólo incluya los cursos ofrecidos durante la primavera de 2006.

```
SELECT CourseNo, Enrollment.OfferNo, Count(*) AS NumStudents
FROM Offering, Enrollment
WHERE Offering.OfferNo = Enrollment.OfferNo
    AND OffYear = 2006 AND OffTerm = 'SPRING'
GROUP BY Enrollment.OfferNo, CourseNo
```

CourseNo	OfferNo	NumStudents
FIN480	7777	3
IS460	9876	7
IS480	5679	6

El ejemplo 4.38 muestra otro problema que incluye enlaces y agrupación. Una parte importante de este problema es la necesidad de la tabla *Student* y de la condición HAVING. Éstos son necesarios ya que la sentencia del problema se refiere a una función agregada que involucra a la tabla *Student*.

**EJEMPLO 4.38****Enlaces, agrupaciones y condiciones de agrupación**

Liste el número de curso, el número de oferta y el promedio de los estudiantes inscritos. Sólo incluya los cursos ofrecidos durante el otoño de 2005 en los que el promedio de los estudiantes inscritos sea mayor a 3.0.

```
SELECT CourseNo, Enrollment.OfferNo, Avg(StdGPA) AS AvgGPA
FROM Student, Offering, Enrollment
WHERE Offering.OfferNo = Enrollment.OfferNo
    AND Enrollment.StdSSN = Student.StdSSN
    AND OffYear = 2005 AND OffTerm = 'FALL'
GROUP BY CourseNo, Enrollment.OfferNo
HAVING Avg(StdGPA) > 3.0
```

CourseNo	OfferNo	AvgGPA
IS320	1234	3.23333330949148
IS320	4321	3.03333334128062

#### 4.5.5 Operadores de conjuntos tradicionales en SQL

En SQL puede utilizar de forma directa los operadores de conjuntos tradicionales con las palabras clave UNION, INTERSECT y EXCEPT. Al igual que en el álgebra relacional, el problema siempre es asegurarse que las tablas sean compatibles con respecto al operador unión. En SQL se puede utilizar la sentencia SELECT para hacer las tablas compatibles incluyendo únicamente las columnas compatibles. Del ejemplo 4.39 al 4.41 se muestran las operaciones de conjuntos hechas sobre los subconjuntos de columnas de las tablas *Faculty* y *Student*. Las columnas han sido renombradas para evitar confusiones.

**EJEMPLO 4.39 Consulta UNION**

Muestre todos los profesores y estudiantes. Sólo muestre las columnas comunes en el resultado.

```
SELECT FacSSN AS SSN, FacFirstName AS FirstName, FacLastName AS
      LastName, FacCity AS City, FacState AS State
   FROM Faculty
UNION
SELECT StdSSN AS SSN, StdFirstName AS FirstName, StdLastName AS
      LastName, StdCity AS City, StdState AS State
   FROM Student
```

SSN	FirstName	LastName	City	State
098765432	LEONARD	VINCE	SEATTLE	WA
123456789	HOMER	WELLS	SEATTLE	WA
124567890	BOB	NORBERT	BOTHELL	WA
234567890	CANDY	KENDALL	TACOMA	WA
345678901	WALLY	KENDALL	SEATTLE	WA
456789012	JOE	ESTRADA	SEATTLE	WA
543210987	VICTORIA	EMMANUEL	BOTHELL	WA
567890123	MARIAH	DODGE	SEATTLE	WA
654321098	LEONARD	FIBON	SEATTLE	WA
678901234	TESS	DODGE	REDMOND	WA
765432109	NICKI	MACON	BELLEVUE	WA
789012345	ROBERTO	MORALES	SEATTLE	WA
876543210	CRISTOPHER	COLAN	SEATTLE	WA
890123456	LUKE	BRAZZI	SEATTLE	WA
901234567	WILLIAM	PILGRIM	BOTHELL	WA
987654321	JULIA	MILLS	SEATTLE	WA

**EJEMPLO 4.40 Consulta INTERSECT  
(Oracle)**

Muestre a los asistentes de profesor y a los catedráticos que son estudiantes. Sólo muestre las columnas comunes en el resultado.

```
SELECT FacSSN AS SSN, FacFirstName AS FirstName, FacLastName AS
      LastName, FacCity AS City, FacState AS State
   FROM Faculty
INTERSECT
SELECT StdSSN AS SSN, StdFirstName AS FirstName,
      StdLastName AS LastName, StdCity AS City,
      StdState AS State
   FROM Student
```

SSN	FirstName	LastName	City	State
876543210	CRISTOPHER	COLAN	SEATTLE	WA

**EJEMPLO 4.41  
(Oracle)****Consulta con diferencia**

Muestre los catedráticos que *no* son estudiantes (sólo catedráticos). Muestre en el resultado sólo las columnas en común. Oracle utiliza la palabra clave MINUS en lugar de la palabra clave EXCEPT utilizada en SQL:2003.

```
SELECT FacSSN AS SSN, FacFirstName AS FirstName, FacLastName AS
      LastName, FacCity AS City, FacState AS State
   FROM Faculty
      MINUS
SELECT StdSSN AS SSN, StdFirstName AS FirstName, StdLastName AS
      LastName, StdCity AS City, StdState AS State
   FROM Student
```

SSN	FirstName	LastName	City	State
098765432	LEONARD	VINCE	SEATTLE	WA
543210987	VICTORIA	EMMANUEL	BOTHELL	WA
654321098	LEONARD	FIBON	SEATTLE	WA
765432109	NICKI	MACON	BELLEVUE	WA
987654321	JULIA	MILLS	SEATTLE	WA

Las filas duplicadas se eliminan por omisión en los resultados de las sentencias de SQL con las palabras clave UNION, INTERSECT y EXCEPT (MINUS). Si quiere conservar las filas duplicadas, utilice la palabra clave ALL después del operador. Por ejemplo, la palabra clave UNION ALL realiza una operación de unión pero no elimina las filas duplicadas.

## 4.6 Sentencias de modificación de SQL

Las sentencias de modificación sirven para agregar filas nuevas (INSERT), modificar columnas de una o más filas (UPDATE), y eliminar una o más filas (DELETE). A pesar de que están bien diseñadas y son poderosas, no se usan tanto como la sentencia SELECT, ya que los formularios de captura de datos facilitan su uso a los usuarios finales.

La sentencia INSERT tiene dos formatos, tal como se muestra en los ejemplos 4.42 y 4.43. En el primer formato sólo se agrega una fila a la vez. Usted especifica los valores para cada columna con la cláusula VALUES. Debe formatear los valores constantes apropiados para cada columna. Consulte la documentación de su DBMS para obtener más detalles sobre constantes específicas, especialmente sobre las cadenas y las constantes de fechas. La especificación de un valor nulo para una columna es algo que tampoco está estandarizado en los DBMS. En algunos sistemas usted simplemente omite el nombre de la columna y el valor. En otros sistemas se usa un símbolo particular para un valor nulo. Claro, usted debe tener cuidado de que la definición de la tabla permita valores nulos para la columna que le interese. De otra manera la sentencia INSERT será rechazada.

**EJEMPLO 4.42****Inserte una sola fila**

Inserte una fila en la tabla *Student* proporcionando los valores de todas las columnas.

```
INSERT INTO Student
      (StdSSN, StdFirstName, StdLastName, StdCity, StdState, StdZip, StdClass,
       StdMajor, StdGPA)
    VALUES ('999999999', 'JOE', 'STUDENT', 'SEATAC', 'WA', '98042-1121', 'FR',
           'IS', 0.0)
```

El segundo formato de la sentencia INSERT soporta la adición de un conjunto de registros, tal como se muestra en el ejemplo 4.43. Es posible especificar y derivar conjuntos de registros utilizando la sentencia SELECT dentro de la sentencia INSERT. Puede utilizar el segundo formato cuando quiera crear tablas temporales para un procesamiento especializado.

#### **EJEMPLO 4.43      Inserte filas múltiples**

Asuma que la nueva tabla *ISStudent* ha sido creada previamente. La tabla *ISStudent* tiene las mismas columnas que *Student*. Esta sentencia INSERT agrega registros de la tabla *Student* a *ISStudent*.

```
INSERT INTO ISStudent
SELECT * FROM Student WHERE StdMajor = 'IS'
```

La sentencia UPDATE permite que uno o más registros se modifiquen, tal como se muestra en los ejemplos 4.44 y 4.45. Se puede modificar cualquier número de columnas, aunque típicamente sólo se puede modificar una columna a la vez. Cuando se quiere modificar la llave primaria, las reglas de actualización de las filas referenciadas quizás no permitan llevar a cabo la operación.

#### **EJEMPLO 4.44      Actualice una sola columna**

Asigne a los miembros del profesorado del departamento MS un 10 por ciento de aumento. Se actualizan cuatro filas.

```
UPDATE Faculty
SET FacSalary = FacSalary * 1.1
WHERE FacDept = 'MS'
```

#### **EJEMPLO 4.45      Actualice varias columnas**

Modifique la carrera y la clase de Homer Wells. Se actualiza una fila.

```
UPDATE Student
SET StdMajor = 'ACCT', StdClass = 'SO'
WHERE StdFirstName = 'HOMER'
AND StdLastName = 'WELLS'
```

La sentencia DELETE permite que se eliminen uno o más registros, tal como se muestra en los ejemplos 4.46 y 4.47. La sentencia DELETE depende de las reglas de las filas referenciadas. Por ejemplo, una fila de *Student* no puede eliminarse si existe una relación con el registro *Enrollment* y la acción de eliminación está restringida.

#### **EJEMPLO 4.46      Elimine las filas seleccionadas**

Elimine todas las especialidades IS de estudiantes avanzados. Se eliminan tres filas.

```
DELETE FROM Student
WHERE StdMajor = 'IS' AND StdClass = 'SR'
```

**EJEMPLO 4.47****Elimine todas las filas de una tabla**

Elimine todas las filas de la tabla *ISStudent*. Este ejemplo asume que la tabla *ISStudent* se creó previamente.

```
DELETE FROM ISStudent
```

Algunas veces es útil para la condición dentro de la cláusula WHERE de la sentencia DELETE hacer referencia a las filas de otra tabla. Microsoft Access soporta el estilo del operador de enlace para combinar las tablas, tal como se muestra en el ejemplo 4.48. *No puede* utilizar el estilo del producto cruz dentro de la sentencia DELETE. El capítulo 9 muestra otra manera de referenciar otras tablas en la sentencia DELETE que la mayoría de los DBMS soportan (incluyendo Access y Oracle).

**EJEMPLO 4.48  
(Access)****Sentencia DELETE con el uso del estilo del operador de enlace**

Elimine los cursos ofrecidos por Leonard Vince. Se eliminan tres filas Offering. Además, esta sentencia elimina las filas relacionadas en la tabla Enrollment, ya que la cláusula ON DELETE está configurada en CASCADE.

```
DELETE Offering.*  
FROM Offering INNER JOIN Faculty  
    ON Offering.FacSSN = Faculty.FacSSN  
WHERE FacFirstName = 'LEONARD'  
    AND FacLastName = 'VINCE'
```

**Reflexión final**

El capítulo 4 introdujo las sentencias fundamentales del estándar de la industria, Lenguaje de Consulta Estructurada (SQL). SQL tiene un amplio alcance que cubre las definiciones, manipulación y control de las bases de datos. Como resultado de un minucioso análisis y compromiso, los grupos de estándares generaron un lenguaje bien diseñado. SQL se ha convertido en un pegamento común que liga la industria de bases de datos, aun cuando no existe un cumplimiento estricto con el estándar.

Este capítulo se enfocó en las partes que más se usan de la sentencia SELECT desde la parte central del estándar SQL:2003. Se mostraron numerosos ejemplos para demostrar las condiciones sobre distintos tipos de datos, expresiones lógicas complejas, tablas con múltiples enlaces, agrupación de tablas con GROUP BY y HAVING, ordenación de tablas y operadores tradicionales de conjuntos. Para facilitar el uso práctico de SQL, se mostraron ejemplos en Oracle y Access, poniendo especial atención en las desviaciones del estándar SQL:2003. Este capítulo también describió de forma breve las sentencias de modificación INSERT, UPDATE y DELETE. Estas sentencias no son tan complejas ni tan utilizadas como la sentencia SELECT.

Este capítulo puso énfasis en dos guías para la solución de problemas con el fin de ayudarle a formular las consultas. El proceso de evaluación conceptual se presentó para demostrar la derivación de los renglones resultantes de las sentencias SELECT que involucran enlaces y agrupaciones. Tal vez encuentre que este proceso de evaluación le ayuda en su aprendizaje inicial de la sentencia SELECT, al tiempo que le proporciona un bosquejo para problemas más retadores. Para ayudar a la formulación de consultas se le proporcionaron tres preguntas de guía. Para resolver un problema debe responder explícita o implícitamente estas preguntas antes de escribir una sentencia SELECT.

La comprensión de las preguntas críticas y del proceso de evaluación conceptual le proporcionará una base sólida para usar bases de datos relacionales. Aun con estas ayudas de formulación de consultas necesitará trabajar con muchos problemas para aprender a formular consultas y usar la sentencia SELECT.

Este capítulo cubrió un subconjunto importante de la sentencia SELECT. Otras partes de la sentencia SELECT que no se cubrieron en este capítulo son los enlace externos, consultas anidadas y problemas de división. El capítulo 9 cubre la formulación de consultas avanzadas y partes adicionales de la sentencia SELECT para que pueda mejorar sus habilidades.

## Revisión de conceptos

- SQL está formado por sentencias de definición de bases de datos (CREATE TABLE, ALTER TABLE, etc.), manipulación de bases de datos (SELECT, INSERT, UPDATE y DELETE) y control de bases de datos (GRANT, REVOKE, etc.).
- El estándar de SQL más reciente se conoce como SQL:2003. Los mayores fabricantes de DBMS respaldan la mayoría de las características de la parte nuclear del estándar, aunque la falta de cumplimiento independiente esconde un cumplimiento estricto con el estándar.
- SELECT es una sentencia compleja. El capítulo 4 cubre las sentencias SELECT con el formato:

```
SELECT <list of column and column expressions>
      FROM <list of tables and join operations>
            WHERE <list of row conditions connected by AND, OR, and NOT>
                  GROUP BY <list of columns>
                        HAVING <list of group conditions connected by AND, OR, and NOT>
                              ORDER BY <list of sorting specifications>
```

- Utilice los operadores estándares de comparación para seleccionar filas:

```
SELECT StdFirstName, StdLastName, StdCity, StdGPA
      FROM Student
            WHERE StdGPA >= 3.7
```

- Una coincidencia inexacta se realiza con el operador LIKE y con los caracteres de búsqueda de patrones:

Oracle y SQL:2003

```
SELECT CourseNo, CrsDesc
      FROM Course
            WHERE CourseNo LIKE 'IS4%'
```

Access

```
SELECT CourseNo, CrsDesc
      FROM Course
            WHERE CourseNo LIKE 'IS4*'!
```

- Utilice BETWEEN . . . AND para comparar fechas:

Oracle

```
SELECT FacFirstName, FacLastName, FacHireDate
      FROM Faculty
            WHERE FacHireDate BETWEEN '1-Jan-1999' AND '31-Dec-2000'
```

Access:

```
SELECT FacFirstName, FacLastName, FacHireDate
  FROM Faculty
 WHERE FacHireDate BETWEEN #1/1/1999# AND #12/31/2000#
```

- Utilice expresiones en la lista de la columna SELECT y en la cláusula WHERE:  
Oracle

```
SELECT FacFirstName, FacLastName, FacCity, FacSalary*1.1 AS
      InflatedSalary, FacHireDate
  FROM Faculty
 WHERE to_number(to_char(FacHireDate, 'YYYY')) > 1999
```

Access

```
SELECT FacFirstName, FacLastName, FacCity, FacSalary*1.1 AS
      InflatedSalary, FacHireDate
  FROM Faculty
 WHERE year(FacHireDate) > 1999
```

- Pruebas para los valores nulos:

```
SELECT OfferNo, CourseNo
  FROM Offering
 WHERE FacSSN IS NULL AND OffTerm = 'SUMMER'
   AND OffYear = 2006
```

- Genere expresiones lógicas complejas con AND y OR:

```
SELECT OfferNo, CourseNo, FacSSN
  FROM Offering
 WHERE (OffTerm = 'FALL' AND OffYear = 2005)
   OR (OffTerm = 'WINTER' AND OffYear = 2006)
```

- Ordene los resultados con la cláusula ORDER BY:

```
SELECT StdGPA, StdFirstName, StdLastName, StdCity, StdState
  FROM Student
 WHERE StdClass = 'JR'
 ORDER BY StdGPA
```

- Elimine los duplicados con la palabra clave DISTINCT:

```
SELECT DISTINCT FacCity, FacState
  FROM Faculty
```

- Califique los nombres de las columnas en las consultas de enlace:

```
SELECT Course.CourseNo, CrsDesc
  FROM Offering, Course
 WHERE OffTerm = 'SPRING' AND OffYear = 2006
   AND Course.CourseNo = Offering.CourseNo
```

- Utilice la cláusula GROUP BY para agrupar las filas:

```
SELECT StdMajor, AVG(StdGPA) AS AvgGpa
  FROM Student
 GROUP BY StdMajor
```

- La cláusula GROUP BY debe estar antes de HAVING:

```
SELECT StdMajor, AVG(StdGPA) AS AvgGpa
  FROM Student
 GROUP BY StdMajor
 HAVING AVG(StdGPA) > 3.1
```

- Utilice WHERE para probar las condiciones de las filas y HAVING para probar las condiciones de los grupos.

```
SELECT StdMajor, AVG(StdGPA) AS AvgGpa
  FROM Student
 WHERE StdClass IN ('JR', 'SR')
 GROUP BY StdMajor
 HAVING AVG(StdGPA) > 3.1
```

- Access no soporta la diferencia entre COUNT(\*) y COUNT (DISTINCT column):

```
SELECT OffYear, COUNT(*) AS NumOfferings, COUNT(DISTINCT CourseNo)
      AS NumCourses
  FROM Offering
 GROUP BY OffYear
```

- Las lecciones del proceso de evaluación conceptual: utilice tablas de ejemplo pequeñas, GROUP BY ocurre después de WHERE, sólo se puede hacer una agrupación por cada sentencia SELECT.
- Preguntas para la generación de consultas: ¿qué tablas?, ¿cómo se combinan?, ¿será una salida de filas o agrupada?
- Enlazando dos o más tablas con el producto cruz y los estilos del operador de enlace (no lo aceptan las versiones de Oracle previas a la 9i):

```
SELECT OfferNo, Offering.CourseNo, CrsUnits, OffDays, OffLocation,
       OffTime
  FROM Faculty, Course, Offering
 WHERE Faculty.FacSSN = Offering.FacSSN
       AND Offering.CourseNo = Course.CourseNo
       AND OffYear = 2005 AND OffTerm = 'FALL'
       AND FacFirstName = 'LEONARD'
       AND FacLastName = 'VINCE'
SELECT OfferNo, Offering.CourseNo, CrsUnits, OffDays, OffLocation, OffTime
  FROM ( Faculty INNER JOIN Offering
           ON Faculty.FacSSN = Offering.FacSSN )
        INNER JOIN Course
           ON Offering.CourseNo = Course.CourseNo
 WHERE OffYear = 2005 AND OffTerm = 'FALL'
       AND FacFirstName = 'LEONARD'
       AND FacLastName = 'VINCE'
```

- Autoenlaces:

```
SELECT Subr.FacSSN, Subr.FacLastName, Subr.FacSalary,
       Supr.FacSSN, Supr.FacLastName, Supr.FacSalary
  FROM Faculty Subr, Faculty Supr
 WHERE Subr.FacSupervisor = Supr.FacSSN
       AND Subr.FacSalary > Supr.FacSalary
```

- Combinación de enlaces y agrupación:

```
SELECT CourseNo, Enrollment.OfferNo, Count(*) AS NumStudents
  FROM Offering, Enrollment
 WHERE Offering.OfferNo = Enrollment.OfferNo
   AND OffYear = 2006 AND OffTerm = 'SPRING'
 GROUP BY Enrollment.OfferNo, CourseNo
```

- Conjunto de operadores tradicionales y compatibilidad con el operador unión:

```
SELECT FacSSN AS SSN, FacLastName AS LastName, FacCity AS City,
       FacState AS State
  FROM Faculty
 UNION
SELECT StdSSN AS SSN, StdLastName AS LastName, StdCity AS City,
       StdState AS State
  FROM Student
```

- Uso de la sentencia INSERT para agregar una o más filas:

```
INSERT INTO Student
  (StdSSN, StdFirstName, StdLastName, StdCity, StdState, StdClass,
   StdMajor, StdGPA)
VALUES ('9999999999', 'JOE', 'STUDENT', 'SEATAC', 'WA', 'FR', 'IS', 0.0)
```

- Uso de la sentencia UPDATE para modificar columnas en una o más filas:

```
UPDATE Faculty
  SET FacSalary = FacSalary * 1.1
 WHERE FacDept = 'MS'
```

- Uso de la sentencia DELETE para eliminar una o más filas:

```
DELETE FROM Student
 WHERE StdMajor = 'IS' AND StdClass = 'SR'
```

- Uso de una operación de enlace dentro de una sentencia DELETE (sólo para Access):

```
DELETE Offering.*
  FROM Offering INNER JOIN Faculty
    ON Offering.FacSSN = Faculty.FacSSN
   WHERE Faculty.FirstName = 'LEONARD' AND Faculty.LastName = 'VINCE'
```

## Preguntas

1. ¿Por qué algunas personas pronuncian SQL como “sequel”?
2. ¿Por qué las sentencias de manipulación de SQL se usan con mayor frecuencia que las sentencias de definición y control?
3. ¿Cuántos niveles tienen los estándares SQL-92, SQL:1999 y SQL:2003?
4. ¿Por qué es importante la prueba de compatibilidad para el estándar SQL?
5. ¿En general, cuál es el grado de compatibilidad entre los mayores fabricantes de DBMS con respecto al estándar SQL:2003?
6. ¿Qué es el SQL individual?
7. ¿Qué es el SQL embebido?
8. ¿Qué es una expresión dentro del contexto de las bases de datos?
9. A partir de los ejemplos y comentarios del capítulo 4, ¿qué partes de la sentencia SELECT no se encuentran disponibles para todos los DBMS?
10. Escriba la regla sobre las cláusulas GROUP BY y HAVING.
11. Escriba la regla sobre las columnas de la instrucción SELECT cuando se usa una cláusula GROUP BY.

12. ¿En qué difiere una condición de fila de una condición de grupo?
13. ¿Por qué las condiciones de las filas se deben colocar en la cláusula WHERE y no en la cláusula HAVING?
14. ¿Por qué la mayoría de los DBMS son indistintos respecto al uso de mayúsculas/minúsculas cuando se intenta hacer coincidir dos cadenas de caracteres?
15. Explique por qué al trabajar con tablas de ejemplo se pueden identificar problemas difíciles.
16. Cuando se trabaja con columnas de fechas, ¿por qué es necesario revisar la documentación de su DBMS?
17. ¿Cómo difieren las coincidencias exactas e inexactas en SQL?
18. ¿Cómo sabe cuándo el resultado de una consulta se relaciona con grupos de filas contrariamente a la relación de filas individuales?
19. ¿Qué tablas pertenecen a la sentencia FROM?
20. Explique el estilo del producto cruz para las operaciones de enlace.
21. Explique el estilo del operador de enlace para las operaciones de enlace.
22. Comente las ventajas y desventajas del producto cruz *versus* el operador de enlace. ¿Necesita conocer tanto el producto cruz como el operador de enlace?
23. ¿Qué es un autoenlace? ¿Cuándo es útil un autoenlace?
24. Proporcione un ejemplo de la sentencia SELECT en el cual se requiera una tabla, incluso cuando la tabla no proporcione las condiciones de prueba o columnas que se muestren en el resultado.
25. ¿Cuál es el requerimiento cuando se usan los operadores de conjuntos tradicionales en una sentencia SELECT?
26. Cuando se combinan enlaces y agrupaciones, ¿conceptualmente qué ocurre primero, los enlaces o las agrupaciones?
27. ¿Cuántas veces ocurre la agrupación en una sentencia SELECT?
28. ¿Por qué se usa la sentencia SELECT con mayor frecuencia que las sentencias de modificación INSERT, UPDATE y DELETE?
29. Proporcione un ejemplo de una sentencia INSERT que pueda insertar varias filas.
30. ¿Cuál es la relación entre la sentencia DELETE y las reglas sobre la eliminación de filas referenciadas?
31. ¿Cuál es la relación entre la sentencia UPDATE y las reglas sobre la actualización de la llave primaria de las filas referenciadas?
32. ¿Cómo difiere COUNT(\*) de COUNT(Column Name)?
33. ¿Cómo difiere COUNT(DISTINCT Column Name) de COUNT(Column Name)?
34. Cuando se mezclan AND y OR en una expresión lógica, ¿por qué es bueno el uso de paréntesis?
35. ¿Cuáles son las lecciones más importantes acerca del proceso conceptual de evaluación?
36. ¿Cuáles son los pasos mentales involucrados en la formulación de consultas?
37. ¿Qué tipo de consultas de enlaces generalmente tienen duplicados en el resultado?
38. ¿Qué pasos mentales del proceso de formulación de consultas se toman en cuenta por el proceso conceptual de evaluación y preguntas críticas?

## Problemas

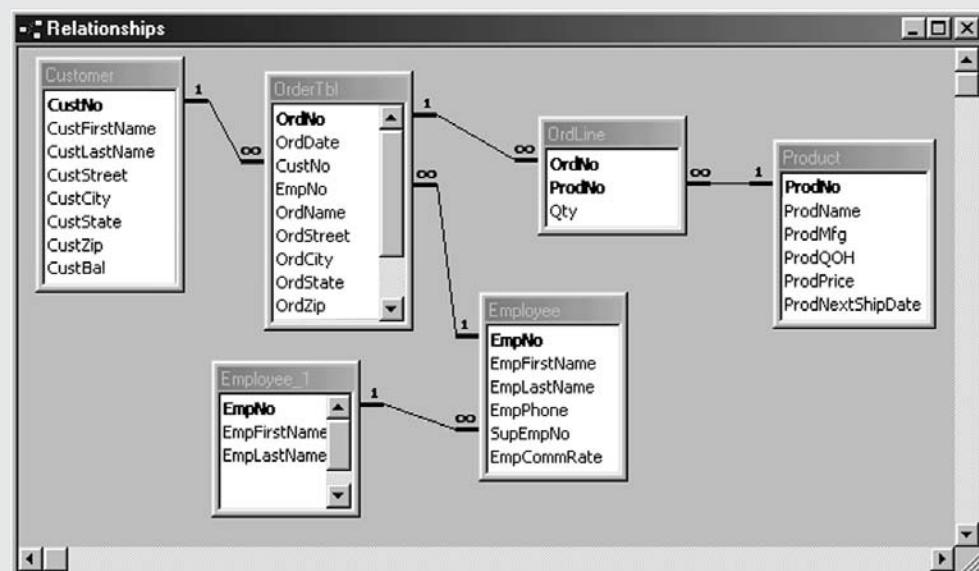
Los problemas usan las tablas de la base de datos de captura de órdenes, una extensión de las tablas de captura de órdenes utilizadas en los problemas del capítulo 3. La tabla 4.P1 lista el significado de cada tabla y la figura 4.P1 muestra la ventana de relaciones de Access. Después del diagrama de relaciones, se muestran los listados de las filas y las sentencias CREATE TABLE de Oracle para cada una de las tablas. Además del resto de la documentación, enseguida se muestran algunas notas sobre la base de datos de captura de órdenes:

- La llave primaria de la tabla *OrdLine* es una combinación de *OrdNo* y *ProdNo*.
- La tabla *Employee* tiene una autorreferencia (unaria) a través de la llave foránea, *SupEmpNo*, el número de empleado del supervisor. En el diagrama de relaciones, la tabla *Employee\_I* es una representación de la relación autorreferenciada, no una tabla real.
- La relación de *OrderTbl* a *OrdLine* realiza eliminaciones y actualizaciones de llaves primarias en cascada de las filas referenciadas. El resto de las relaciones restringen la eliminación y actualización de las llaves primarias de las filas referenciadas, en caso de que existan las filas.

**TABLA 4.P1**  
Tablas de la base de datos de captura de órdenes

Nombre de la tabla	Descripción
Customer	Lista de clientes que han colocado órdenes
OrderTbl	Contiene la parte del encabezado de una orden; las órdenes hechas por Internet no tienen empleado
Employee	Lista de empleados que han registrado las órdenes
OrdLine	Contiene la parte del detalle de una orden
Product	Lista de productos que se pueden ordenar

**FIGURA 4.P1**  
Ventana de relaciones de la base de datos de captura de órdenes



### Customer

CustNo	CustFirstName	CustLastName	CustStreet	CustCity	CustState	CustZip	CustBal
C0954327	Sheri	Gordon	336 Hill St.	Littleton	CO	80129-5543	\$230.00
C1010398	Jim	Glussman	1432 E. Ravenna	Denver	CO	80111-0033	\$200.00
C2388597	Beth	Taylor	2396 Rafter Rd	Seattle	WA	98103-1121	\$500.00
C3340959	Betty	Wise	4334 153rd NW	Seattle	WA	98178-3311	\$200.00
C3499503	Bob	Mann	1190 Lorraine Cir.	Monroe	WA	98013-1095	\$0.00
C8543321	Ron	Thompson	789 122nd St.	Renton	WA	98666-1289	\$85.00
C8574932	Wally	Jones	411 Webber Ave.	Seattle	WA	98105-1093	\$1,500.00
C8654390	Candy	Kendall	456 Pine St.	Seattle	WA	98105-3345	\$50.00
C9128574	Jerry	Wyatt	16212 123rd Ct.	Denver	CO	80222-0022	\$100.00
C9403348	Mike	Boren	642 Crest Ave.	Englewood	CO	80113-5431	\$0.00
C9432910	Larry	Styles	9825 S. Crest Lane	Bellevue	WA	98104-2211	\$250.00
C9543029	Sharon	Johnson	1223 Meyer Way	Fife	WA	98222-1123	\$856.00
C9549302	Todd	Hayes	1400 NW 88th	Lynnwood	WA	98036-2244	\$0.00
C9857432	Homer	Wells	123 Main St.	Seattle	WA	98105-4322	\$500.00
C9865874	Mary	Hill	206 McCaffrey	Littleton	CO	80129-5543	\$150.00
C9943201	Harry	Sanders	1280 S. Hill Rd.	Fife	WA	98222-2258	\$1,000.00

**OrderTbl**

OrdNo	OrdDate	CustNo	EmpNo	OrdName	OrdStreet	OrdCity	OrdState	OrdZip
O1116324	01/23/2007	C0954327	E8544399	Sheri Gordon	336 Hill St.	Littleton	CO	80129-5543
O1231231	01/23/2007	C9432910	E9954302	Larry Styles	9825 S. Crest Lane	Bellevue	WA	98104-2211
O1241518	02/10/2007	C9549302		Todd Hayes	1400 NW 88th	Lynnwood	WA	98036-2244
O1455122	01/09/2007	C8574932	E9345771	Wally Jones	411 Webber Ave.	Seattle	WA	98105-1093
O1579999	01/05/2007	C9543029	E8544399	Tom Johnson	1632 Ocean Dr.	Des Moines	WA	98222-1123
O1615141	01/23/2007	C8654390	E8544399	Candy Kendall	456 Pine St.	Seattle	WA	98105-3345
O1656777	02/11/2007	C8543321		Ron Thompson	789 122nd St.	Renton	WA	98666-1289
O2233457	01/12/2007	C2388597	E9884325	Beth Taylor	2396 Rafter Rd	Seattle	WA	98103-1121
O2334661	01/14/2007	C0954327	E1329594	Mrs. Ruth Gordon	233 S. 166th	Seattle	WA	98011
O3252629	01/23/2007	C9403348	E9954302	Mike Boren	642 Crest Ave.	Englewood	CO	80113-5431
O3331222	01/13/2007	C1010398		Jim Glussman	1432 E. Ravenna	Denver	CO	80111-0033
O3377543	01/15/2007	C9128574	E8843211	Jerry Wyatt	16212 123rd Ct.	Denver	CO	80222-0022
O4714645	01/11/2007	C2388597	E1329594	Beth Taylor	2396 Rafter Rd	Seattle	WA	98103-1121
O5511365	01/22/2007	C3340959	E9884325	Betty White	4334 153rd NW	Seattle	WA	98178-3311
O6565656	01/20/2007	C9865874	E8843211	Mr. Jack Sibley	166 E. 344th	Renton	WA	98006-5543
O7847172	01/23/2007	C9943201		Harry Sanders	1280 S. Hill Rd.	Fife	WA	98222-2258
O7959898	02/19/2007	C8543321	E8544399	Ron Thompson	789 122nd St.	Renton	WA	98666-1289
O7989497	01/16/2007	C3499503	E9345771	Bob Mann	1190 Lorraine Cir.	Monroe	WA	98013-1095
O8979495	01/23/2007	C9865874		HelenSibley	206 McCaffrey	Renton	WA	98006-5543
O9919699	02/11/2007	C9857432	E9954302	Homer Wells	123 Main St.	Seattle	WA	98105-4322

**Employee**

EmpNo	EmpFirstName	EmpLastName	EmpPhone	EmpEmail	SupEmpNo	EmpCommRate
E1329594	Landi	Santos	(303) 789-1234	LSantos@bigco.com	E8843211	0.02
E8544399	Joe	Jenkins	(303) 221-9875	JJenkins@bigco.com	E8843211	0.02
E8843211	Amy	Tang	(303) 556-4321	ATang@bigco.com	E9884325	0.04
E9345771	Colin	White	(303) 221-4453	CWhite@bigco.com	E9884325	0.04
E9884325	Thomas	Johnson	(303) 556-9987	TJohnson@bigco.com		0.05
E9954302	Mary	Hill	(303) 556-9871	MHill@bigco.com	E8843211	0.02
E9973110	Theresa	Beck	(720) 320-2234	TBeck@bigco.com	E9884325	

**Product**

ProdNo	ProdName	ProdMfg	ProdQOH	ProdPrice	ProdNextShipDate
P0036566	17 inch Color Monitor	ColorMeg, Inc.	12	\$169.00	2/20/2007
P0036577	19 inch Color Monitor	ColorMeg, Inc.	10	\$319.00	2/20/2007
P1114590	R3000 Color Laser Printer	Connex	5	\$699.00	1/22/2007
P1412138	10 Foot Printer Cable	Ethlite	100	\$12.00	
P1445671	8-Outlet Surge Protector	Intersafe	33	\$14.99	
P1556678	CVP Ink Jet Color Printer	Connex	8	\$99.00	1/22/2007
P3455443	Color Ink Jet Cartridge	Connex	24	\$38.00	1/22/2007
P4200344	36-Bit Color Scanner	UV Components	16	\$199.99	1/29/2007
P6677900	Black Ink Jet Cartridge	Connex	44	\$25.69	
P9995676	Battery Back-up System	Cybercx	12	\$89.00	2/1/2007

**OrdLine**

OrdNo	ProdNo	Qty
O1116324	P1445671	1
O1231231	P0036566	1
O1231231	P1445671	1
O1241518	P0036577	1
O1455122	P4200344	1
O1579999	P1556678	1
O1579999	P6677900	1
O1579999	P9995676	1
O1615141	P0036566	1
O1615141	P1445671	1
O1615141	P4200344	1
O1656777	P1445671	1
O1656777	P1556678	1
O2233457	P0036577	1
O2233457	P1445671	1
O2334661	P0036566	1
O2334661	P1412138	1
O2334661	P1556678	1
O3252629	P4200344	1
O3252629	P9995676	1
O3331222	P1412138	1
O3331222	P1556678	1
O3331222	P3455443	1
O3377543	P1445671	1
O3377543	P9995676	1
O4714645	P0036566	1
O4714645	P9995676	1
O5511365	P1412138	1
O5511365	P1445671	1
O5511365	P1556678	1
O5511365	P3455443	1
O5511365	P6677900	1
O6565656	P0036566	10
O7847172	P1556678	1
O7847172	P6677900	1
O7959898	P1412138	5
O7959898	P1556678	5
O7959898	P3455443	5
O7959898	P6677900	5
O7989497	P1114590	2
O7989497	P1412138	2
O7989497	P1445671	3
O8979495	P1114590	1
O8979495	P1412138	1
O8979495	P1445671	1
O9919699	P0036577	1
O9919699	P1114590	1
O9919699	P4200344	1

```
CREATE TABLE Customer
(
    CustNo           CHAR(8),
    CustFirstName   VARCHAR2(20) CONSTRAINT CustFirstNameRequired NOT NULL,
    CustLastName    VARCHAR2(30) CONSTRAINT CustLastNameRequired NOT NULL,
    CustStreet      VARCHAR2(50),
    CustCity        VARCHAR2(30),
    CustState       CHAR(2),
    CustZip         CHAR(10),
    CustBal         DECIMAL(12,2) DEFAULT 0,
    CONSTRAINT PKCustomer PRIMARY KEY (CustNo)
)
```

```
CREATE TABLE OrderTbl
(
    OrdNo          CHAR(8),
    OrdDate        DATE    CONSTRAINT OrdDateRequired NOT NULL,
    CustNo         CHAR(8) CONSTRAINT CustNoRequired NOT NULL,
    EmpNo          CHAR(8),
    OrdName        VARCHAR2(50),
    OrdStreet      VARCHAR2(50),
    OrdCity        VARCHAR2(30),
    OrdState       CHAR(2),
    OrdZip         CHAR(10),
    CONSTRAINT PKOrderTbl PRIMARY KEY (OrdNo),
    CONSTRAINT FKCustomer FOREIGN KEY (CustNo) REFERENCES Customer,
    CONSTRAINT FKEmployee FOREIGN KEY (EmpNo) REFERENCES Employee
)
```

```
CREATE TABLE OrdLine
(
    OrdNo          CHAR(8),
    ProdNo         CHAR(8),
    Qty            INTEGER DEFAULT 1,
    CONSTRAINT PKOrdLine PRIMARY KEY (OrdNo, ProdNo),
    CONSTRAINT FKOrdNo FOREIGN KEY (OrdNo) REFERENCES OrderTbl
        ON DELETE CASCADE,
    CONSTRAINT FKProdNo FOREIGN KEY (ProdNo) REFERENCES Product
)
```

```
CREATE TABLE Employee
(
    EmpNo          CHAR(8),
    EmpFirstName   VARCHAR2(20) CONSTRAINT EmpFirstNameRequired NOT NULL,
    EmpLastName    VARCHAR2(30) CONSTRAINT EmpLastNameRequired NOT NULL,
    EmpPhone       CHAR(15),
    EmpEMail       VARCHAR(50) CONSTRAINT EmpEMailRequired NOT NULL,
    SupEmpNo       CHAR(8),
    EmpCommRate    DECIMAL(3,3),
    CONSTRAINT PKEmployee PRIMARY KEY (EmpNo),
    CONSTRAINT UNIQUEEMail UNIQUE(EmpEMail),
    CONSTRAINT FKSupEmpNo FOREIGN KEY (SupEmpNo) REFERENCES Employee
)
```

```

CREATE TABLE Product
(
    ProdNo           CHAR(8),
    ProdName        VARCHAR2(50) CONSTRAINT ProdNameRequired NOT NULL,
    ProdMfg          VARCHAR2(20) CONSTRAINT ProdMfgRequired NOT NULL,
    ProdQOH          INTEGER DEFAULT 0,
    ProdPrice        DECIMAL(12,2) DEFAULT 0,
    ProdNextShipDate DATE,
    CONSTRAINT PKProduct PRIMARY KEY (ProdNo)
)

```

### Parte 1: SELECT

1. Liste el número de cliente; nombre, apellido y saldo de los clientes.
2. Liste el número de cliente; nombre, apellido y el saldo de los clientes que viven en Colorado (*CustState* es CO).
3. Liste todas las columnas de la tabla *Product* para los productos que cuestan más de 50 dólares. Ordene el resultado por el fabricante (*ProdMfg*) y el nombre del producto.
4. Liste el número y la fecha de la orden y nombre del envío (*OrdName*) para las órdenes enviadas a direcciones en Denver o Englewood.
5. Liste el número de cliente, el nombre, apellido, ciudad y saldo de los clientes que viven en Denver y tengan un saldo mayor a 150 dólares, o que vivan en Seattle con un saldo mayor a 300.
6. Liste las ciudades y estados en donde se hayan colocado las órdenes. Elimine los duplicados del resultado.
7. Liste todas las columnas de la tabla *OrderTbl* de las órdenes por Internet levantadas en enero de 2007. Una orden de Internet no tiene asociado un número de empleado.
8. Liste todas las columnas de la tabla *OrderTbl* de las órdenes levantadas en febrero de 2007. Una orden telefónica tiene asociado un número de empleado.
9. Liste todas las columnas de la tabla *Product* que contengan las palabras *Ink Jet* en el nombre del producto.
10. Liste el número y la fecha de la orden; número, nombre y apellido del cliente para las órdenes levantadas después del 23 de enero de 2007, enviadas a residentes de Washington.
11. Liste el número y la fecha de la orden; número, nombre y apellido del cliente para las órdenes levantadas en enero de 2007, enviadas a residentes de Colorado.
12. Liste el número y la fecha de la orden; número, nombre y apellido del cliente para las órdenes colocadas en enero de 2007, levantadas por clientes de Colorado (*CustState*) pero enviadas a residentes de Washington (*OrdState*).
13. Liste el número, nombre y apellido del cliente y saldo de los clientes de Washington que hayan colocado una o más órdenes en febrero de 2007. Elimine las filas duplicadas del resultado.
14. Liste el número y la fecha de la orden; número, nombre y apellido del cliente; número, nombre y apellido del empleado para las órdenes levantadas en enero de 2007, colocadas por clientes de Colorado.
15. Liste el número de empleado; nombre, apellido y teléfonos de los empleados que hayan levantado órdenes en enero de 2007 para clientes con saldos mayores a 300 dólares. Elimine las filas duplicadas del resultado.
16. Liste el número de producto; nombre y precio de los productos ordenados por el cliente número C0954327 en enero de 2007. Elimine los productos duplicados del resultado.
17. Liste el número, nombre y apellido del cliente; número y fecha de la orden; número, nombre y apellido del empleado; número y nombre del producto y costo de la orden (*OrdLine.Qty \* ProdPrice*) para productos ordenados el 23 de enero de 2007, en los que el costo de la orden excede 150 dólares.
18. Liste el saldo promedio de los clientes por ciudad. Incluya únicamente a los clientes que viven en el estado de Washington (WA).

19. Liste el promedio del saldo de los clientes por ciudad y código postal (los primeros cinco dígitos del código postal). Incluya únicamente a los residentes del estado de Washington (WA). En Microsoft Access la expresión `left(CustZip, 5)` regresa los primeros cinco dígitos del código postal. En Oracle la expresión `substr(Custzip, 1, 5)` regresa los primeros cinco dígitos.
20. Liste el saldo promedio y el número de clientes por ciudad. Sólo incluya a los residentes del estado de Washington (WA). Elimine del resultado las ciudades con menos de dos clientes.
21. Liste el número de códigos postales únicos y el saldo promedio de los clientes por ciudad. Sólo incluya a los residentes del estado de Washington (WA). Elimine del resultado las ciudades en las que el saldo promedio sea menor a 100 dólares. En Microsoft Access la expresión `left(CustZip, 5)` regresa los primeros cinco dígitos del código postal. En Oracle la expresión `substr(Custzip, 1, 5)` regresa los primeros cinco dígitos. (*Nota:* Este problema requiere de dos sentencias SELECT en Access SQL o de una consulta anidada en la cláusula FROM. Revise el capítulo 9.)
22. Liste el número de orden y el monto total de las órdenes colocadas el 23 de enero de 2007. El monto total de una orden es la suma de la cantidad por el precio del producto de cada producto que se encuentra en la orden.
23. Liste el número de orden, fecha de la orden, nombre y apellido del cliente y monto total de las órdenes colocadas el 23 de enero de 2007. El monto total de una orden es la suma de la cantidad por el precio del producto para cada producto que se encuentra en la orden.
24. Liste el número, nombre y apellido del cliente; la suma de la cantidad de productos ordenados y el monto total de la orden (suma del precio del producto por la cantidad) de las órdenes colocadas en enero de 2007. Sólo incluya los productos cuyo nombre contenga Ink Jet o Laser. Sólo incluya a los clientes que hayan ordenado más de dos productos Ink Jet o Laser durante enero de 2007.
25. Liste el número y nombre del producto; suma de la cantidad de productos ordenados y monto total de la orden (suma del precio del producto por la cantidad) de las órdenes colocadas durante enero de 2007. Sólo incluya las órdenes que hayan pedido más de 5 productos durante enero de 2007. Ordene el resultado en orden descendente con respecto al monto total de la orden.
26. Liste el número y la fecha de la orden; el número, nombre y apellido del cliente; el estado del cliente y el estado al que se envía (*OrdState*) en el que el estado del cliente es distinto al estado donde se envía.
27. Liste el número, nombre y apellido del empleado; la tasa de comisión, el nombre y apellido del empleado que supervisa y la tasa de comisión del supervisor.
28. Liste el número, nombre y apellido del empleado; y el monto total de las comisiones de las órdenes levantadas en enero de 2007. El monto de la comisión es la suma del monto en dólares de los productos ordenados por la tasa de comisión del empleado.
29. Liste la unión de clientes y receptores de órdenes. Incluya en el resultado el nombre, calle, ciudad, estado y código postal. Necesita usar la función de concatenación para combinar el nombre y el apellido para que se puedan comparar con el nombre del receptor. En Access SQL, el símbolo & es la función de concatenación. En Oracle SQL el símbolo || es la función de concatenación.
30. Liste el nombre y apellido de los clientes que tengan el mismo nombre y apellido de algún empleado.
31. Liste el número de empleado y el nombre y apellido de los subordinados de segundo nivel (subordinados de los subordinados) del empleado Thomas Johnson.
32. Liste el número de empleado y el nombre y apellido de los subordinados de primer y segundo nivel del empleado Thomas Johnson. Para distinguir entre los niveles de subordinación, incluya una columna calculada con el nivel del subordinado (1 o 2).
33. Utilizando una combinación de los estilos del operador de enlace y del producto cruz, liste los nombres y apellidos de los clientes que colocaron órdenes registradas por Amy Tang. Elimine las filas duplicadas del resultado. Observe que el estilo del operador de enlace sólo se incluye en versiones de Oracle 9i o superiores.
34. Utilizando el estilo del operador de enlace, liste el nombre del producto y el precio de todos los productos ordenados por Beth Taylor durante enero de 2007. Elimine las filas duplicadas del resultado.
35. Para los clientes de Colorado, calcule el número de órdenes colocadas durante enero de 2007. El resultado debe incluir el número de cliente, apellido y número de órdenes colocadas durante enero de 2007.
36. Para los clientes de Colorado, calcule el número de órdenes colocadas durante enero de 2007 donde la orden incluya productos hechos por Connex. El resultado debe incluir el número de cliente, el apellido y el número de órdenes colocadas durante enero de 2007.

37. Para cada empleado con una tasa de comisiones menor al 0.04, calcule el número de órdenes registradas durante enero de 2007. El resultado debe incluir el número y apellido del empleado y número de órdenes registradas.
38. Para cada empleado con una tasa de comisiones mayor al 0.03, calcule el total de comisiones ganadas por las órdenes registradas durante enero de 2007. La comisión total ganada es el monto total de la orden por la tasa de comisión. El resultado debe incluir el número y apellido del empleado y la comisión total ganada.
39. Liste el monto total de todas las órdenes por mes de 2007. El resultado debe incluir el mes y el monto total de todas las órdenes de cada mes. El monto total de una orden individual es la suma de la cantidad por el precio de cada producto de la orden. En Access, el número del mes se puede extraer con la función **Month** tomando una fecha como argumento. Usted puede mostrar el mes utilizando la función **MonthName** aplicándola a un número de mes. En Oracle, la función *to\_char(OrdDate, 'M')* extrae el número de mes de *OrdDate*. Utilizando “MON” en lugar de “M” se extrae la abreviación de tres dígitos en lugar del número del mes.
40. Liste la comisión total ganada por cada empleado en cada mes durante 2007. El resultado debe incluir el mes, número y apellido del empleado y el monto total de comisiones ganadas en ese mes. El monto de la comisión para un empleado individual es la suma en dólares del monto de los productos ordenados por la tasa de comisión del empleado. Ordene el resultado por mes de forma ascendente y por monto total de comisión de forma descendente. En Access, el número del mes se puede extraer con la función **Month** tomando una fecha como argumento. Usted puede mostrar el mes utilizando la función **MonthName** y aplicándola a un número de mes. En Oracle, la función *to\_char(OrdDate, 'M')* extrae el número de mes de *OrdDate*. Utilizando “MON” en lugar de “M” se extrae la abreviación de tres dígitos en vez del número del mes.

## Parte 2: Sentencias INSERT, UPDATE y DELETE

1. Agréguese a sí mismo como una nueva fila de la tabla *Customer*.
2. Agregue a su compañero, mejor amigo u otro, como fila nueva en la tabla *Employee*.
3. Agregue una fila nueva de *OrderTbl* siendo usted un cliente de la persona del problema 2 (parte 2) como empleado y elija los valores que desee para el resto de las columnas de la tabla *OrderTbl*.
4. Agregue dos filas en la tabla *OrdLine* correspondiente a la fila *OrderTbl* insertada en el problema 3 (parte 2).
5. Aumente el precio en 10 por ciento de los productos que contengan las palabras *Ink Jet*.
6. Modifique la dirección (calle, ciudad y código postal) de la fila nueva que se agregó en el problema 1 (parte 2).
7. Identifique una orden que respete las reglas acerca de la eliminación de filas referenciadas para eliminar las filas insertadas en los problemas 1 al 4 (parte 2).
8. Elimine las filas nuevas de la primera tabla listada del problema 7 (parte 2).
9. Elimine las nuevas filas de la segunda tabla listada en el problema 7 (parte 2).
10. Elimine las nuevas filas de las tablas restantes del problema 7 (parte 2).

## Referencias para ampliar su estudio

Existen muchos libros sobre SQL que van desde la cobertura básica, avanzada y específica para un producto. Un buen resumen de libros de SQL se puede encontrar en [www.ocelot.ca/books.htm](http://www.ocelot.ca/books.htm).

El sitio *DBAZine* ([www.dbazine.com](http://www.dbazine.com)) y la *DevX.com Database Zone* ([www.devx.com](http://www.devx.com)) tienen bastantes consejos acerca de la formulación de consultas y SQL. Para obtener consejos específicos de algún producto, el sitio *Advisor.com* ([www.advisor.com](http://www.advisor.com)) muestra artículos técnicos de Microsoft SQL Server y Microsoft Access. La documentación de Oracle se puede encontrar en el sitio *Oracle Technet* ([www.oracle.com/technology](http://www.oracle.com/technology)).

## Apéndice 4.A

### Resumen de sintaxis de SQL:2003

Este apéndice resume la sintaxis de SQL:2003 para las sentencias SELECT, INSERT, UPDATE y DELETE presentadas en este capítulo. La sintaxis se limita a la estructura de sentencias simplificada presentada en este capítulo. La sintaxis más compleja se presenta en la parte 5 de este libro. Las formas usadas en la notación de la sintaxis son idénticas a las utilizadas en el capítulo 3.

#### Sintaxis SELECT simplificada

```

<Select-Statement>: { <Simple-Select> | <Set-Select> }
    [ ORDER BY <Sort-Specification>* ]

<Simple-Select>:
    SELECT [ DISTINCT ] <Column-Specification>*
        FROM <Table-Specification>*
        [ WHERE <Row-Condition> ]
        [ GROUP BY ColumnName* ]
        [ HAVING <Group-Condition> ]

<Column-Specification>: { <Column-List> | <Column-Item> }

<Column-List>: { * | TableName.* }
    — * is a literal here not a syntax symbol

<Column-Item>: <Column-Expression> [ AS ColumnName ]

<Column-Expression>:
    { <Scalar-Expression> | <Aggregate-Expression> }

<Scalar-Expression>:
    { <Scalar-Item> |
        <Scalar-Item> <Arith-Operator> <Scalar-Item> }

<Scalar-Item>:
    { [ TableName.]ColumnName |
        Constant |
        FunctionName [ (Argument*) ] |
        <Scalar-Expression> |
        ( <Scalar-Expression> ) }

<Arith-Operator>: { + | - | * | / }
    — * and + are literals here not syntax symbols

<Aggregate-Expression>:
    { SUM ( {<Scalar-Expression> | DISTINCT ColumnName} ) |
        AVG ( {<Scalar-Expression> | DISTINCT ColumnName} ) |

```

```

MIN ( <Scalar-Expression> ) |
MAX ( <Scalar-Expression> ) |
COUNT ( [ DISTINCT ] ColumnName ) |
COUNT ( * ) } — * is a literal symbol here, not a special syntax symbol

<Table-Specification>: { <Simple-Table> |
                           <Join-Operation> }

<Simple-Table>: TableName [ [ AS ] AliasName ]

<Join-Operation>:
    { <Simple-Table> [INNER] JOIN <Simple-Table>
      ON <Join-Condition> |
      { <Simple-Table> | <Join-Operation> } [INNER] JOIN
        { <Simple-Table> | <Join-Operation> }
        ON <Join-Condition> |
        ( <Join-Operation> ) }

<Join-Condition>: { <Simple-Join-Condition> |
                     <Compound-Join-Condition> }

<Simple-Join-Condition>:
    <Scalar-Expression> <Comparison-Operator>
    <Scalar-Expression>

<Compound-Join-Condition>:
    { NOT <Join-Condition> |
      <Join-Condition> AND <Join-Condition> |
      <Join-Condition> OR <Join-Condition> |
      ( <Join-Condition> ) }

<Comparison-Operator>: { = | < | > | <= | >= | <> }

<Row-Condition>:
    { <Simple-Condition> | <Compound-Condition> }

<Simple-Condition>:
    { <Scalar-Expression> <Comparison-Operator>
      <Scalar-Expression> |
      <Scalar-Expression> [ NOT ] IN ( Constant* ) |
      <Scalar-Expression> BETWEEN <Scalar-Expression> AND
      <Scalar-Expression> |
      <Scalar-Expression> IS [NOT] NULL |
      ColumnName [ NOT ] LIKE StringPattern }

<Compound-Condition>:
    { NOT <Row-Condition> |
      <Row-Condition> AND <Row-Condition> |
      <Row-Condition> OR <Row-Condition> |
      ( <Row-Condition> ) }

```

```

<Group-Condition>:
    { <Simple-Group-Condition> | <Compound-Group-Condition> }

<Simple-Group-Condition>:— permits both scalar and aggregate expressions
    { <Column-Expression> ComparisonOperator
        <Column-Expression> |
        <Column-Expression> [ NOT ] IN ( Constant* ) |
        <Column-Expression> BETWEEN <Column-Expression> AND
        <Column-Expression> |
        <Column-Expression> IS [NOT] NULL |
        ColumnName [ NOT ] LIKE StringPattern }

<Compound-Group-Condition>:
    { NOT <Group-Condition> |
        <Group-Condition> AND <Group-Condition> |
        <Group-Condition> OR <Group-Condition> |
        ( <Group-Condition> ) }

<Sort-Specification>:
    { ColumnName | ColumnNumber } [ { ASC | DESC } ]

<Set-Select>:
    { <Simple-Select> | <Set-Select> } <Set-Operator>
    { <Simple-Select> | <Set-Select> }

<Set-Operator>: { UNION | INTERSECT | EXCEPT } [ ALL ]

```

## Sintaxis INSERT

```

INSERT INTO TableName ( ColumnName* )
    VALUES ( Constant* )

INSERT INTO TableName [ ( ColumnName* ) ]
    <Simple-Select>

```

## Sintaxis UPDATE

```

UPDATE TableName
    SET <Column-Assignment>*
    [ WHERE <Row-Condition> ]

<Column-Assignment>: ColumnName = <Scalar-Expression>

```

## Sintaxis DELETE

```
DELETE FROM TableName
[ WHERE <Row-Condition> ]
```

### Apéndice 4.B

## Diferencias de sintaxis entre los principales productos DBMS

La tabla 4B.1 resume las diferencias entre Microsoft Access (versiones 1997 a 2003), Oracle 8i a 10g, Microsoft SQL Server e IBM DB2. Las diferencias incluyen las partes de la sentencia SELECT presentada en este capítulo.

**TABLA 4.B1 Diferencias de sintaxis SELECT entre los principales productos DBMS**

Elemento/Producto	Oracle 8i, 9i, 10g	Access 97/2000/2002/2003			DB2
		MS SQL Server 2000			
Caracteres para búsqueda de patrón	%,_	* , ? aunque los caracteres % y _ se pueden usar en las versiones 2002/2003 al cambiar a modo de consultas	%,_		%, _
Diferencia entre mayúsculas/minúsculas en la búsqueda de cadenas de caracteres	Sí	No	Sí	Sí	
Constantes de fechas	Encierra entre comillas sencillas	Encierra con los símbolos #	Encierra entre comillas sencillas	Encierra entre comillas sencillas	
Símbolo de desigualdad	<>	<>	!=	<>	
Estilo del operador de enlace	No en 8i, Sí en 9i y 10g	Sí	Sí	Sí	
Operaciones de diferencia	Palabra clave MINUS	No soportado	No soportado	Palabra clave EXCEPT	



# Parte 3

## Modelado de datos

---

Los capítulos de la parte 3 se dedican al desarrollo del modelado de datos utilizando el modelo entidad-relación para proporcionar las habilidades necesarias para el diseño conceptual de bases de datos. En el capítulo 5 se presenta la notación de la pata de cuervo (*crow's foot*) del modelo de entidad-relación y explica las reglas del diagrama para evitar sus errores más comunes. El capítulo 6 enfatiza la práctica del modelado de datos con la exposición de problemas y presenta las reglas para transformar los diagramas de entidad-relación (ERD) en tablas relacionales. El capítulo 6 explica las transformaciones del diseño y los errores más comunes del diseño para agudizar las habilidades del modelado de datos.

---

**Capítulo 5.** Comprensión de los diagramas de entidad-relación

**Capítulo 6.** Desarrollo de modelado de datos para bases de datos de negocios



# Capítulo

# 5

---

# Compreensión de los diagramas de entidad-relación

## Objetivos de aprendizaje

Este capítulo explica la notación de los diagramas de entidad-relación como prerequisito para el uso de los diagramas de entidad-relación en el proceso de desarrollo de base de datos. Al finalizar este capítulo, el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Conocer los símbolos y el vocabulario de la notación de la pata de cuervo para los diagramas de entidad-relación.
- Utilizar los símbolos de cardinalidad para representar las relaciones 1-1, 1-M y M-N.
- Comparar la notación de pata de cuervo con la representación de tablas relacionales.
- Comprender importantes patrones de relación.
- Utilizar la generalización de jerarquías para representar tipos similares de entidad.
- Detectar los errores de las notaciones en un diagrama de entidad-relación.
- Comprender la representación de las reglas de negocios en un diagrama de entidad-relación.
- Apreciar la diversidad de notaciones para los diagramas de entidad-relación.

## Panorama general

---

En el capítulo 2 se proporcionó una amplia presentación acerca del proceso de desarrollo de bases de datos. Usted aprendió acerca de las relaciones entre el desarrollo de la base de datos y el desarrollo de sistemas de información, las fases del desarrollo de bases de datos y el tipo de habilidades requeridas para dominarlas. Este capítulo presenta la notación de los diagramas de entidad-relación para sentar las bases del uso de los diagramas de entidad-relación en el proceso de desarrollo de bases de datos. Para ampliar sus habilidades en el diseño de bases de datos, el capítulo 6 describe el proceso del uso de los diagramas de entidad-relación para desarrollar los modelos de datos para las bases de datos de negocios.

Para convertirse en un buen modelador de datos necesita comprender la notación de los diagramas de entidad-relación y aplicar la notación a los problemas de mayor complejidad. Para ayudarle a elaborar una notación, este capítulo presenta los símbolos usados en los diagramas de entidad-relación y compara los diagramas de entidad-relación con los diagramas de bases de datos relacionales que estudió en los capítulos anteriores. Este capítulo se enfoca de una manera

más profunda en las relaciones, la parte más relevante de los diagramas de entidad-relación. Aprenderá a identificar las dependencias, los patrones de relación y la equivalencia entre dos tipos de relaciones. Finalmente, usted aprenderá a representar similitudes entre entidades utilizando la generalización de jerarquías.

Para proporcionar un mayor entendimiento de la notación de la pata de cuervo, se muestra la representación de las reglas de negocios y las reglas de los diagramas. Para proporcionar un enfoque organizacional en los diagramas de entidad-relación, este capítulo ofrece una representación formal e informal de reglas de negocios en un diagrama de entidad-relación. Para ayudarlo a comprender correctamente la notación de la pata de cuervo, este capítulo presenta la consistencia y las reglas completas, así como su uso en el ER Assistant.

Dada la gran cantidad de notaciones de entidad-relación, quizás no tenga la oportunidad de utilizar la notación de la pata de cuervo exactamente como se muestra en los capítulos 5 y 6. Para prepararlo en la comprensión de otras notaciones, el capítulo concluye con la presentación de variaciones del diagrama, que incluye la notación del diagrama de clases de la notación del modelado unificado, una de las alternativas más populares al modelo entidad-relación.

Este capítulo proporciona las habilidades básicas del modelado de datos para que le permita comprender la notación de los diagramas de entidad-relación. Para aplicar el modelado de datos como parte del proceso de desarrollo de bases de datos, debe estudiar el capítulo 6 en la parte del desarrollo de los modelos de datos para las bases de datos de negocios. El capítulo 6 pone énfasis en las habilidades para la solución de problemas al generar diseños alternativos, mapear la sentencia de un problema y justificar las decisiones del diseño. Con los antecedentes que se proporcionan en ambos capítulos, estará preparado para realizar el modelado de datos basado en los casos de estudio y en las bases de datos para organizaciones de tamaño medio.

## 5.1 Introducción a los diagramas de entidad-relación

---

**tipo de entidad**  
una colección de entidades de interés (personas, lugares, eventos o cosas) representadas por un rectángulo en un diagrama de entidad-relación.

**atributo**  
una propiedad de un tipo de entidad o relación. Cada atributo tiene un tipo de dato que define el tipo de valores y operaciones permitidas sobre dicho atributo.

**relación**  
una asociación nombrada entre los tipos de entidades. Una relación representa una asociación en dos sentidos o bidireccional entre entidades. La mayoría de las relaciones involucran dos distintos tipos de entidad.

Para lograr un entendimiento básico de los diagramas de entidad-relación (ERD) se requiere de un estudio cuidadoso. Esta sección introduce la notación de la pata de cuervo (*crow's foot*) para ERD, una notación popular respaldada por muchas herramientas CASE. Para comenzar, esta sección inicia con los símbolos básicos de los tipos de entidades, relaciones y atributos; explica las cardinalidades y su apariencia en la notación de pata de cuervo. Esta sección concluye con la comparación entre la notación de pata de cuervo y los diagramas relacionales de base de datos. Si estudió el modelado de datos antes que las bases de datos relacionales, puede omitir la última parte de esta sección.

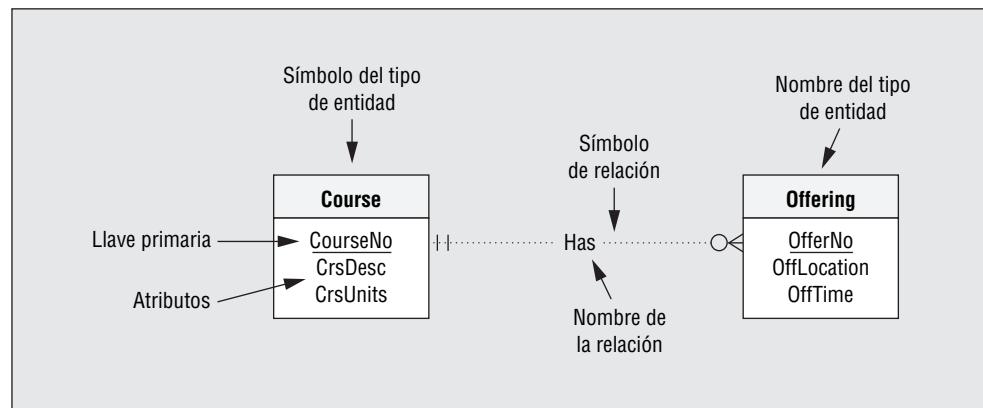
### 5.1.1 Símbolos básicos

Los ERD tienen tres elementos básicos: tipos de entidad, relaciones y atributos. Los tipos de entidad son colecciones de cosas de interés (entidades) en una aplicación. Los tipos de entidad representan colecciones de cosas físicas, como libros, gente y lugares, y también eventos como pagos. Una entidad es un miembro o instancia de un tipo de entidad. Las entidades se identifican únicamente para poder rastreárlas a través de los procesos de negocios. Por ejemplo, los clientes tienen una identificación única para respaldar el procesamiento de órdenes, el embarque y los procesos de las garantías de los productos. En la notación de la pata de cuervo, al igual que en la mayoría de otras notaciones, los rectángulos representan tipos de entidad. En la figura 5.1, el tipo de entidad *Course* representa el conjunto de cursos en la base de datos.

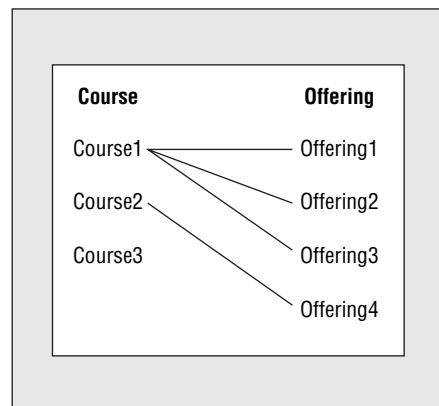
Los atributos son propiedades de los tipos de entidad o relaciones. Un tipo de entidad debe tener la llave primaria al igual que otros atributos descriptivos. Los atributos se muestran dentro de un rectángulo que representa el tipo de entidad-relación. Si existen muchos atributos, se pueden eliminar para listarlos en una página separada. Algunas herramientas de dibujo de ERD muestran los atributos en una vista ampliada, separada del resto del diagrama. Un(os) atributo(s) subrayado(s) indica(n) que es una llave primaria dentro del tipo de entidad.

A las relaciones se les llama asociaciones entre tipos de entidad. En la notación de pata de cuervo, los nombres de las relaciones aparecen en la línea que conecta los tipos de entidades incluidas en la relación. En la figura 5.1, la relación *Has* muestra que los tipos de entidad *Course*

**FIGURA 5.1**  
Diagrama de entidad-relación que ilustra los símbolos básicos



**FIGURA 5.2**  
Diagrama de instancias para la relación *Has*



y *Offering* están directamente relacionados. Las relaciones almacenan asociaciones en ambas direcciones. Por ejemplo, la relación *Has* muestra los cursos ofrecidos para cierto curso y el curso asociado para un curso impartido. La relación *Has* es binaria ya que incluye dos tipos de entidad. La sección 5.2 muestra los ejemplos de relaciones más complejas que incluyen sólo un tipo diferente de entidad (relaciones unitarias) y más de dos tipos de entidades (relaciones M-way).

En un sentido poco estricto, los ERD tienen un lenguaje natural de correspondencia. Los tipos de entidades pueden corresponder al sustantivo y las relaciones a verbos o frases de preposiciones que conectan los sustantivos. En este sentido, uno puede leer el diagrama de entidad-relación como una colección de sentencias. Por ejemplo, el ERD de la figura 5.1 se puede leer como “el curso tiene ofertas”. Observe que existe una dirección implícita en cada relación. En la otra dirección, se podría escribir, “la oferta está dada para un curso”. En la práctica, es una buena idea usar verbos activos en lugar de verbos pasivos para las relaciones. Por lo tanto, se prefiere *Has* como el nombre de la relación. Debe utilizar la correspondencia del lenguaje natural como una guía más que como una regla estricta. Para los ERD grandes no siempre encontrará una buena correspondencia con el lenguaje natural para todas las partes de los diagramas.

#### cardinalidad

una restricción sobre el número de entidades que participan en una relación. En un ERD, el número mínimo y máximo de cardinalidades se especifican para las dos direcciones de la relación.

#### 5.1.2 Cardinalidad de las relaciones

Las cardinalidades restringen el número de objetos que participan en una relación. Para ilustrar el significado de las cardinalidades, es útil un diagrama de instancias. La figura 5.2 muestra un conjunto de cursos ({Course1, Course2, Course3}), un conjunto de cursos ofrecidos ({Offering1, Offering2, Offering3, Offering4}), y conexiones entre los dos conjuntos. En la figura 5.2, Course1 se relaciona con Offering1, Offering2 y Offering3, mientras que Course2 se relaciona con Offering4 y Course3 no se relaciona con ninguna de las entidades *Offering*. Por lo mismo,

Offering1 se relaciona con Course1; Offering2 se relaciona con Course1; Offering3 se relaciona con Course1 y Offering4 se relaciona con Course2. A partir de este diagrama de instancias, podemos concluir que cada oferta se relaciona exactamente con un curso. En el otro sentido, cada curso se relaciona con cero o más ofertas.

#### *Representación de pata de cuervo (crow's foot) de las cardinalidades*

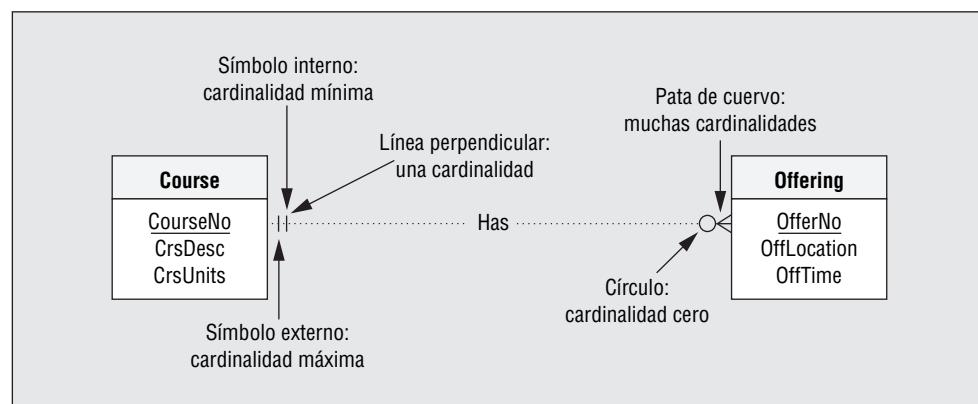
La notación de la pata de cuervo usa tres símbolos para representar las cardinalidades. El símbolo de la pata de cuervo (dos líneas en diagonal y una línea recta) representa muchas entidades relacionadas (cero o más). En la figura 5.3, el símbolo de la pata de cuervo cerca del tipo de entidad *Offering* significa que un curso se puede relacionar con muchos de los cursos ofrecidos. El círculo significa una cardinalidad de cero, mientras que una línea perpendicular a la línea de la relación significa una cardinalidad de uno.

Para ilustrar el mínimo y el máximo de cardinalidades, los símbolos de cardinalidad se colocan a un lado de cada tipo de entidad en una relación. El símbolo de cardinalidad mínima aparece cercano al nombre de la relación, mientras que el símbolo de cardinalidad máxima aparece cercano al tipo de entidad. En la figura 5.3, un curso se relaciona con un mínimo de cero cursos ofrecidos (el círculo en la posición interna) y un máximo de muchos cursos ofrecidos (la pata de cuervo en la posición externa). De forma similar, un curso ofrecido se relaciona con exactamente un curso (uno y sólo uno), tal como se muestra por las líneas verticales de las posiciones interna y externa.

#### **dependencia de la existencia**

una entidad no puede existir a menos que exista otra entidad relacionada. Una relación obligatoria crea una dependencia de la existencia.

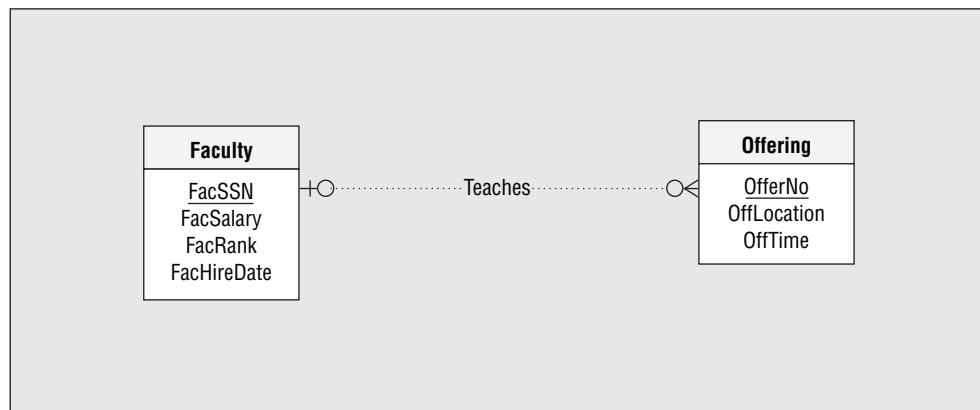
**FIGURA 5.3**  
Diagrama entidad-relación con la notación de las cardinalidades



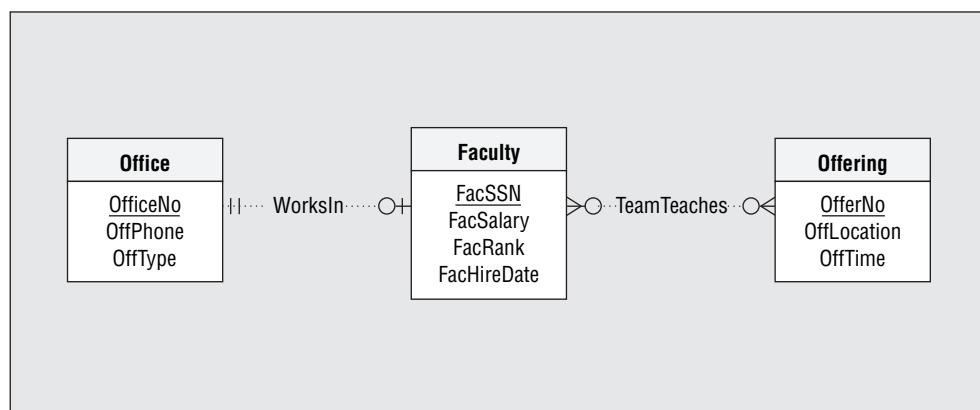
**TABLA 5.1**  
Resumen de la clasificación de cardinalidades

Clasificación	Restricciones de las cardinalidades
Obligatoria	Cardinalidad mínima $\geq 1$
Opcional	Cardinalidad mínima = 0
Funcional o de valor único	Cardinalidad máxima = 1
1-M	Cardinalidad máxima = 1 en una dirección y cardinalidad máxima $> 1$ en la otra dirección
M-N	Cardinalidad máxima $> 1$ en ambas direcciones
1-1	Cardinalidad máxima = 1 en ambas direcciones

**FIGURA 5.4**  
Relaciones opcionales para ambos tipos de entidad



**FIGURA 5.5**  
Ejemplos de relaciones M-N y 1-1



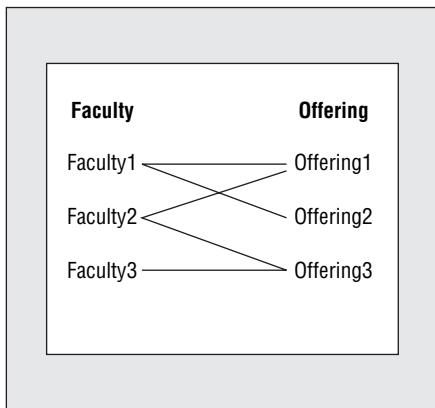
puede almacenarse sin una entidad *Course* relacionada. En contraste, una cardinalidad mínima de cero indica una relación opcional. Por ejemplo, la relación *Has* es opcional para el tipo de entidad *Course*, ya que una entidad *Course* se puede almacenar sin estar relacionada con una entidad *Offering*. La figura 5.4 muestra que la relación *Teaches* es opcional para los dos tipos de entidad.

La tabla 5.1 también muestra diversas clasificaciones para las cardinalidades máximas. Una cardinalidad máxima de uno significa que la relación es de un solo valor o funcional. Por ejemplo, las relaciones *Has* y *Teaches* son funcionales para *Offering*, ya que una entidad *Offering* se puede relacionar con máximo una entidad *Course* y una entidad *Faculty*. La palabra *función* proviene de las matemáticas, donde una función proporciona un valor. Una relación que tienen una cardinalidad máxima de uno en una dirección y más de una (muchas) en la otra, se llama relación 1-M (léase uno-a-muchos). Las relaciones *Has* y *Teaches* son 1-M.

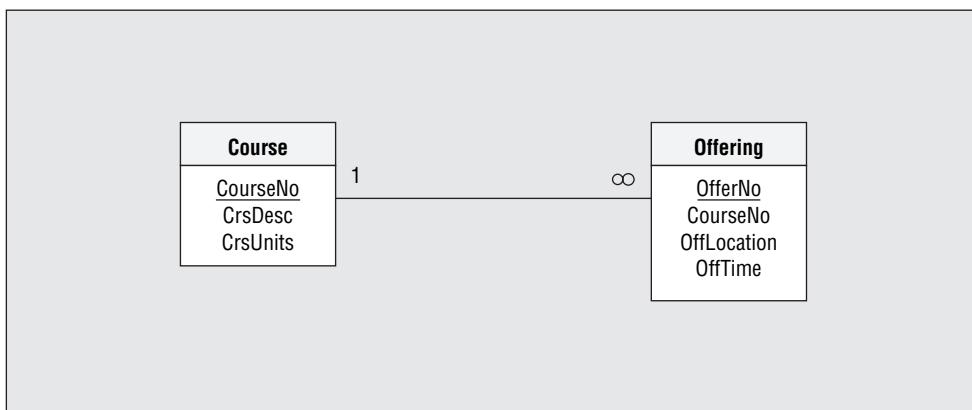
De forma similar, una relación que tiene una cardinalidad máxima mayor a 1 en ambas direcciones se llama relación M-N (muchos-a-muchos). En la figura 5.5, la relación *TeamTeaches* permite que varios profesores se unan para enseñar el mismo curso ofertado, tal como se muestra en el diagrama de instancias de la figura 5.6. Las relaciones M-N son comunes en las bases de datos de negocios para representar la conexión entre partes y proveedores, autores y libros, y habilidades y empleados. Por ejemplo, una parte puede ser entregada por muchos proveedores y un proveedor puede entregar muchas partes.

Menos comunes aún son las relaciones 1-1, en las que la cardinalidad máxima es igual a uno en ambas direcciones. Por ejemplo, la relación *WorksIn* de la figura 5.5 permite que a un catedrático se le asigne una oficina y que una oficina sea ocupada a lo máximo por un sólo catedrático.

**FIGURA 5.6**  
Diagrama de instan-  
cias para la relación  
M-N *TeamTeaches*



**FIGURA 5.7**  
Diagrama de la base  
de datos relacional  
para el ejemplo  
*Course-Offering*



### 5.1.3 Comparación con los diagramas de bases de datos relacionales

Para terminar esta sección comparemos la notación de la figura 5.3 con los diagramas de la base de datos relacional (de Microsoft Access) con los que está familiarizado. Es fácil confundirse entre las dos notaciones. Algunas de las más grandes diferencias se listan a continuación.<sup>1</sup> Para ayudarle a visualizar estas diferencias, la figura 5.7 muestra un diagrama de base de datos relacional para el ejemplo *Course-Offering*.

1. Los diagramas de bases de datos relacionales no usan nombres para las relaciones. En su lugar, las llaves foráneas representan relaciones. La notación ERD no usa llaves foráneas. Por ejemplo, *Offering.CourseNo* es una columna de la figura 5.7, pero no un atributo de la figura 5.3.
2. Los diagramas de bases de datos relacionales sólo muestran las cardinalidades máximas.
3. Algunas notaciones ERD (incluyendo la notación de la pata de cuervo) permiten que tanto los tipos de entidad como las relaciones tengan atributos. Los diagramas de bases de datos relacionales sólo permiten que las tablas tengan columnas.
4. Los diagramas de bases de datos relacionales permiten una relación entre dos tablas. Algunas notaciones ERD (aunque no en la notación de la pata de cuervo) permiten que las relaciones M-way incluyan más de dos tipos de entidad. La siguiente sección muestra la manera de representar las relaciones M-way con la notación de la pata de cuervo.
5. En algunas notaciones ERD (aunque no en la notación de la pata de cuervo), la posición de las cardinalidades es inversa.

<sup>1</sup> El capítulo 6 presenta las reglas de conversión que describen las diferencias de forma más precisa.

## 5.2 Comprensión de las relaciones

Esta sección explora la notación entidad-relación con mayor detalle, examinando importantes aspectos de las relaciones. La primera subsección describe la identificación de dependencias, un tipo especializado de existencia de dependencias. La segunda subsección describe tres patrones importantes de las relaciones: (1) relaciones con atributos, (2) relaciones con referencias a sí mismas, y (3) tipos de entidad asociativas que representan relaciones multiformas (M-way). La última subsección describe una equivalencia importante entre las relaciones M-N y 1-M.

### 5.2.1 Identificación de dependencias (entidades débiles y relación identifiable)

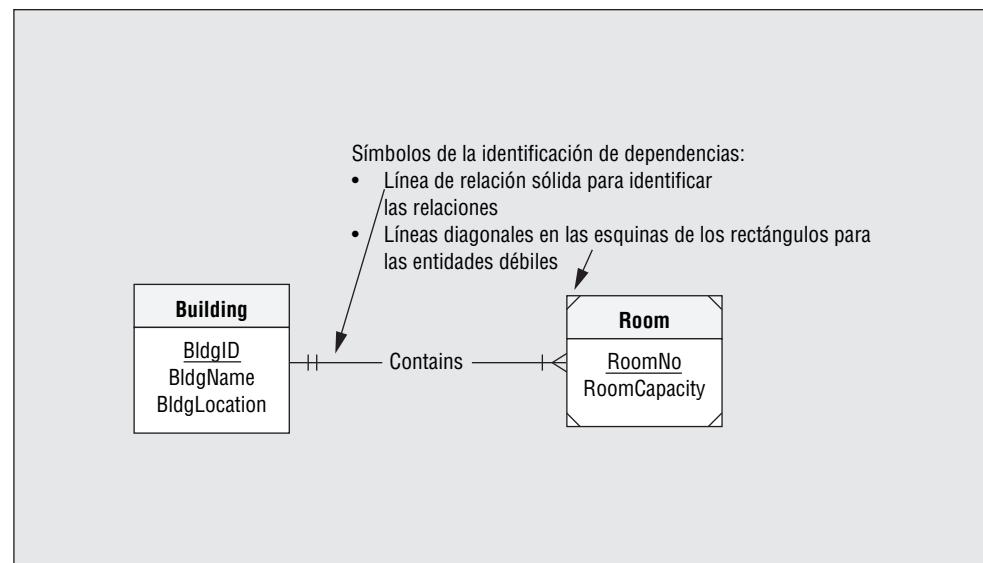
En un ERD, algunos tipos de entidad pueden no tener su propia llave primaria. Los tipos de entidad que no tienen su propia llave primaria deben pedir prestada una parte (o toda) su llave primaria a otros tipos de entidad. A los tipos de entidad que piden prestada una parte o toda la llave primaria se les conoce como entidades débiles. La relación o relaciones que ofrecen componentes de la llave primaria se conocen como relaciones identificables. Por ende, la identificación de dependencias incluye a una entidad débil y a una o más relaciones identificables.

La identificación de dependencias ocurre porque algunas entidades se relacionan demasiado con otras. Por ejemplo, una habitación no tiene una identidad separada de su edificio, ya que una habitación físicamente se encuentra dentro de un edificio. Usted puede ubicar una habitación únicamente proporcionando el identificador del edificio asociado. En el ERD para edificios y habitaciones (figura 5.8), el tipo de entidad *Room* es dependiente del tipo de entidad *Building* en la relación *Contains*. Una línea de relación sólida indica una relación identifiable. Para las entidades débiles, el atributo subrayado (si es que existe) es parte de la llave primaria, pero no es toda la llave primaria. Por ende, la llave primaria de *room* es una combinación de *BldgID* y *RoomNo*. Como ejemplo adicional, la figura 5.9 ilustra una identificación de dependencias que incluye a la entidad débil *State* y la relación identifiable *Holds*.

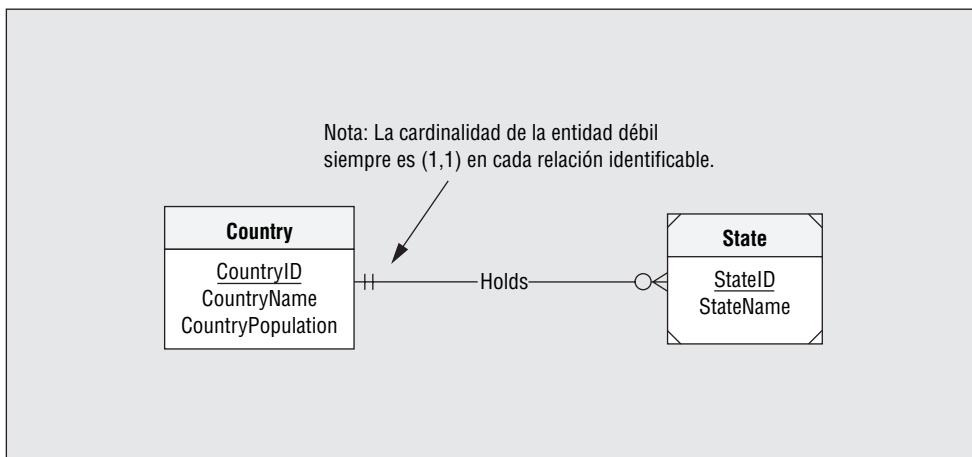
La identificación de dependencias es un tipo especializado de existencia de dependencias. Recuerde que un tipo de entidad existente-dependiente tiene una relación obligatoria (cardinalidad mínima de 1). Las entidades débiles dependen de la existencia de las relaciones identificables. Además de la dependencia de existencia, una entidad débil pide prestada al menos una parte para su llave primaria. Dada la existencia de la dependencia y de haber pedido prestada la llave primaria, las cardinalidades mínima y máxima de una entidad débil son siempre 1.

**entidad débil**  
un tipo de entidad que pide prestada toda o parte de su llave primaria de otro tipo de entidad. Las relaciones identificables señalan los tipos de entidad que proporcionan los componentes de la llave primaria prestada.

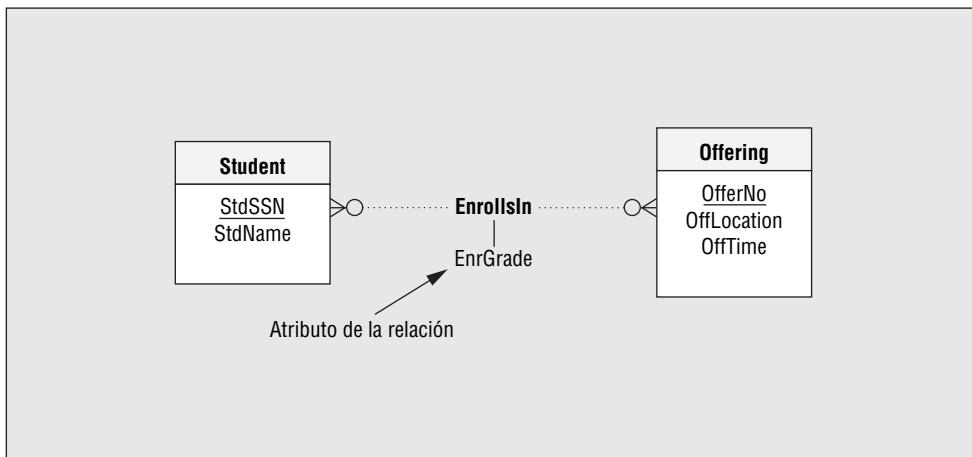
**FIGURA 5.8**  
Ejemplo de la identificación de dependencias



**FIGURA 5.9**  
Ejemplo adicional de la identificación de dependencias



**FIGURA 5.10**  
Relación M-N con un atributo



La siguiente sección muestra otros ejemplos adicionales de la identificación de dependencias en la descripción de los tipos de entidades asociativas y de las relaciones M-way. El uso de la identificación de dependencias es necesaria para los tipos de entidad asociativas.

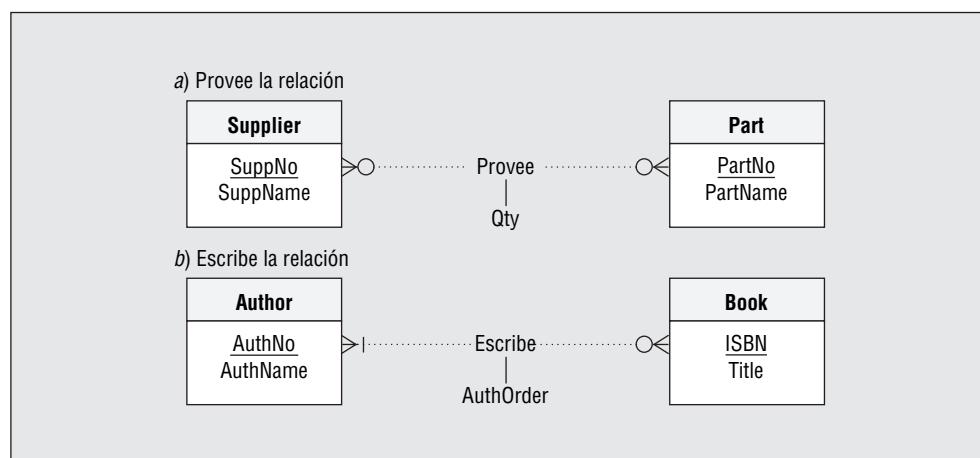
### 5.2.2 Patrones de relación

Esta sección describe tres patrones para las relaciones que pudiese encontrar en los esfuerzos de desarrollo de bases de datos: (1) relaciones M-N con atributos, (2) relaciones autorreferenciadas (unitarias), y (3) tipos de entidad asociativa que representan relaciones M-way. Aunque estos patrones de relaciones no dominan los ERD, cuando suceden tienen su importancia. Necesita estudiar estos patrones de forma cuidadosa para aplicarlos en los esfuerzos de desarrollo de bases de datos.

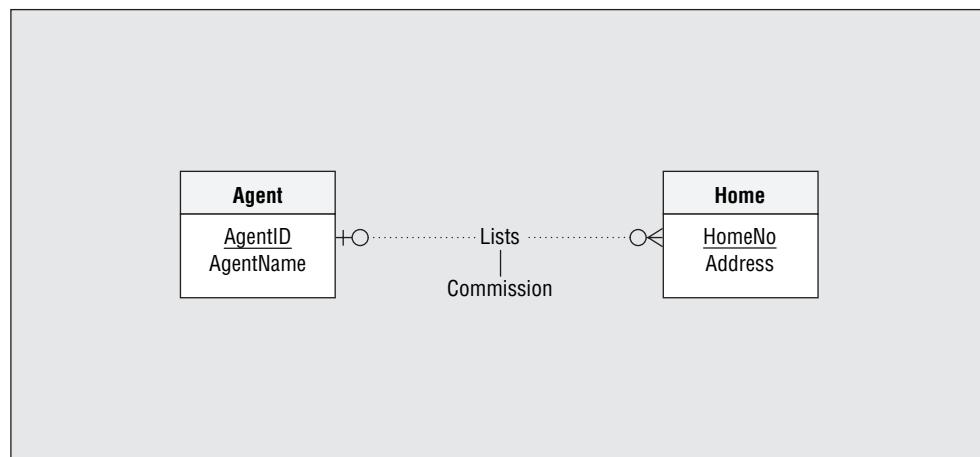
#### Relaciones M-N con atributos

Como se describió brevemente en la sección 5.1, las relaciones pueden tener atributos. Esta situación típicamente ocurre con las relaciones M-N. En una relación M-N, los atributos se asocian con la combinación de los tipos de entidad, y no sólo con uno de los tipos de entidad. Si un atributo está asociado con un sólo tipo de entidad, debe ser parte de dicho tipo de entidad, no de la relación. Las figuras 5.10 y 5.11 ilustran las relaciones M-N con atributos. En la figura 5.10, el atributo *EnrGrade* se asocia con la combinación de un estudiante y un curso ofrecido, pero no con alguno de ellos por sí solo. Por ejemplo, la relación *EnrollsIn* registra el hecho de que el estudiante con número de seguridad social 123-77-9993 tiene una calificación de 3.5 en

**FIGURA 5.11**  
Relaciones M-N adicionales con atributos



**FIGURA 5.12**  
Relación 1-M con un atributo



el curso ofrecido con el número de oferta 1 256. En la figura 5.11a), el atributo *Qty* representa la cantidad de una parte proveída por un determinado proveedor. En la figura 5.11b), el atributo *AuthOrder* representa la orden en la cual el nombre del autor aparece en el título del libro. Para reducir las descripciones de un diagrama grande, puede que no se muestren los atributos de las relaciones.

Las relaciones 1-M también pueden tener atributos, pero las relaciones 1-M con atributos son mucho menos comunes que las relaciones M-N con atributos. En la figura 5.12, el atributo *Commission* se asocia con la relación *Lists*, no con el tipo de entidad *Agent* o *Home*. Una casa sólo tendrá una comisión si un agente lo lista. Típicamente, las relaciones 1-M con atributos son opcionales para los tipos de entidad hijos. La relación *Lists* es opcional para el tipo de entidad *Home*.

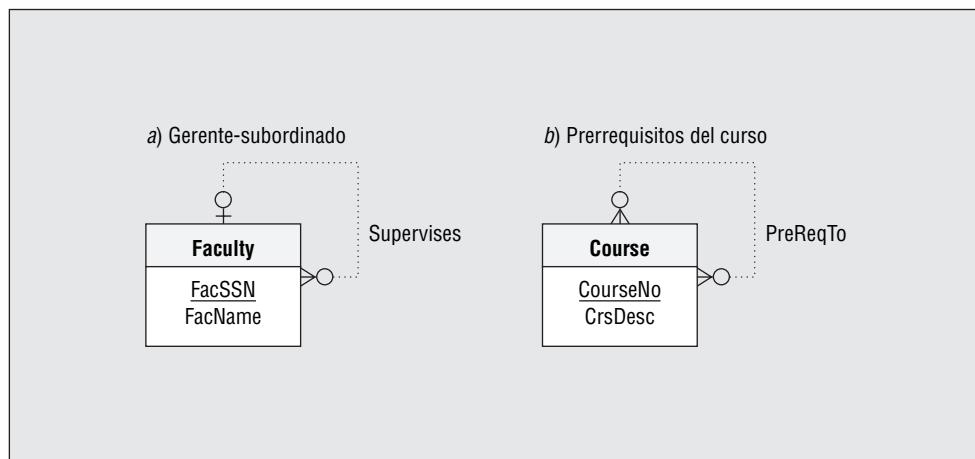
#### *Relaciones autorreferenciadas (unitarias)*

#### **relación autorreferenciada**

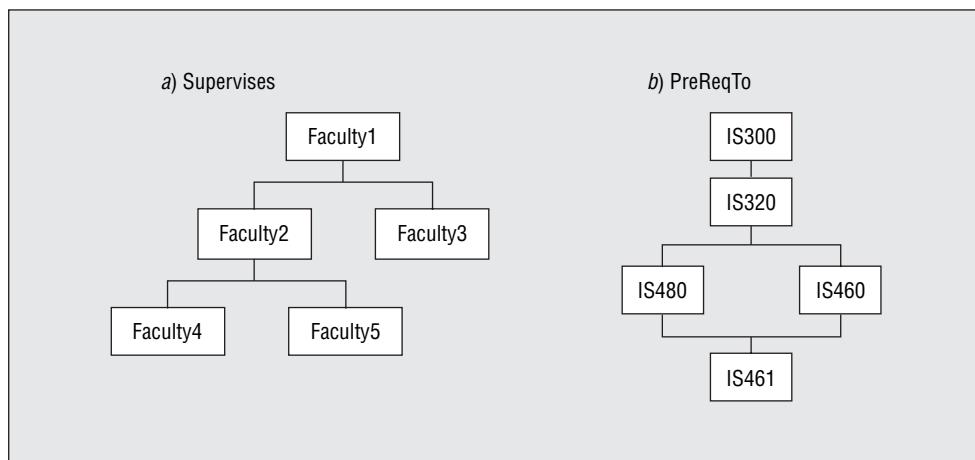
una relación que incluye el mismo tipo de entidad. Las relaciones autorreferenciadas representan asociaciones entre los miembros del mismo conjunto.

Una relación autorreferenciada incluye conexiones entre los miembros del mismo conjunto. A las relaciones autorreferenciadas algunas veces se les denomina relaciones reflexivas, ya que son como el reflejo en un espejo. La figura 5.13 despliega dos relaciones autorreferenciadas que incluyen los tipos de entidad *Faculty* y *Course*. Ambas relaciones incluyen dos tipos de entidad que son los mismos (*Faculty* para *Supervises* y *Course* para *PreReqTo*). Estas relaciones ilustran conceptos importantes en la base de datos de la universidad. La relación *Supervises* ilustra un organigrama, mientras que la relación *PreReqTo* ilustra las dependencias de los cursos que pueden afectar al plan de cursos de un estudiante.

**FIGURA 5.13**  
Ejemplos de relaciones autorreferenciales (unitaria)



**FIGURA 5.14**  
Diagramas de instancias para relaciones autorreferenciadas



Para las relaciones autorreferenciadas es importante diferenciar entre las relaciones 1-M y M-N. Un diagrama de instancias puede ayudarle a comprender la diferencia. La figura 5.14a) muestra un diagrama de instancias para la relación *Supervises*. Observe que cada catedrático puede tener al menos un superior. Por ejemplo, **Faculty2** y **Faculty3** tienen a **Faculty1** como superior. Por lo tanto, *Supervises* es una relación 1-M, ya que cada catedrático puede tener al menos un supervisor. En contraste, no existe ninguna restricción en el diagrama de instancias para la relación *PreReqTo* (figura 5.14b). Por ejemplo, tanto **IS480** como **IS460** son prerequisitos para **IS461**. Por lo tanto, *PreReqTo* es una relación M-N, ya que un curso puede ser prerequisito de muchos cursos, y un curso puede tener muchos prerequisitos.

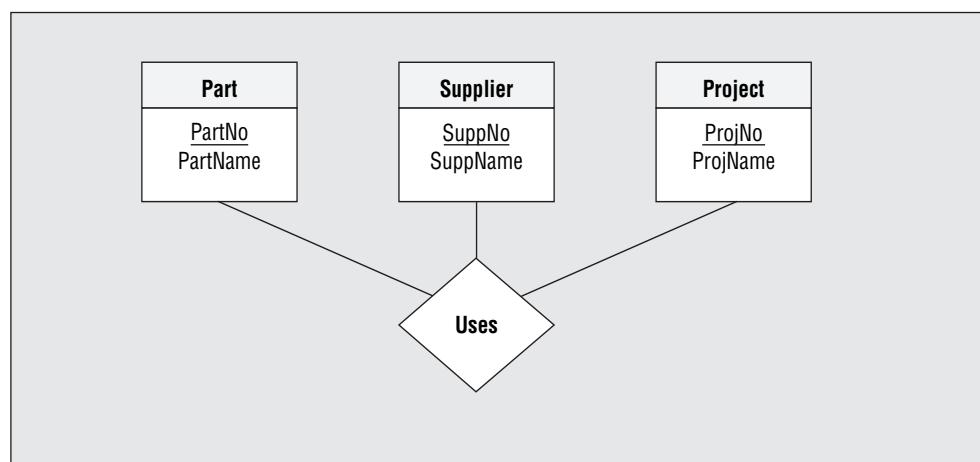
Las relaciones autorreferenciadas suceden con frecuencia en los negocios. Cualquier dato que pueda visualizarse como en la figura 5.14 se puede representar como una relación autorreferenciada. Ejemplos típicos incluyen cuadros jerárquicos de cuentas contables, árboles genealógicos, diseños de partes y rutas de transportación. En estos ejemplos, las relaciones que hacen referencia a sí mismas son una parte importante de la base de datos.

Existe otro aspecto de las relaciones autorreferenciadas, y es que en ocasiones no se necesita una relación autorreferenciada. Por ejemplo, si usted sólo quiere saber si un empleado es un supervisor, no es necesaria una relación autorreferenciada. En su lugar, se puede usar un atributo para indicar si un empleado es un supervisor.

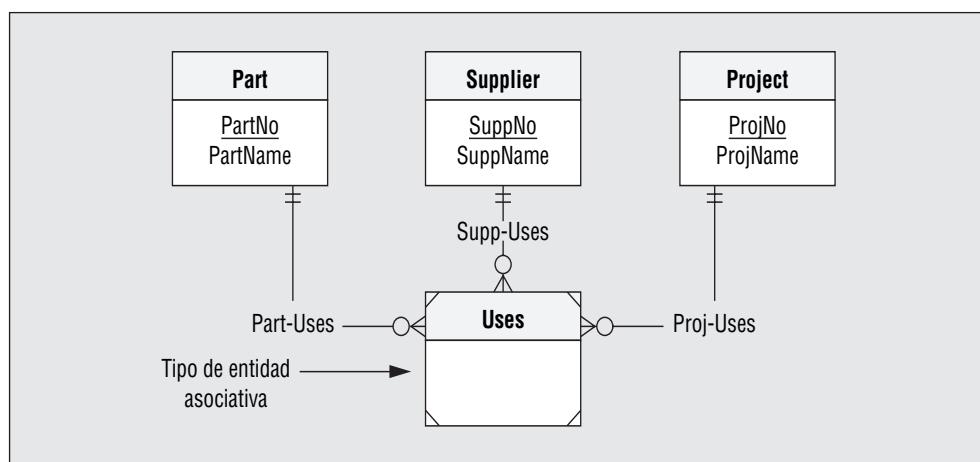
#### *Tipos de entidades asociativas que representan relaciones multiforma (M-Way)*

Algunas notaciones ERD respaldan las relaciones que incluyen más de dos tipos de entidades, conocidas como relaciones M-way (multiforma), en donde la M significa más de dos. Por ejem-

**FIGURA 5.15**  
Relaciones M-forma (ternarias) que usan la notación Chen



**FIGURA 5.16**  
Tipo de entidad asociativa para representar una relación ternaria



En este ejemplo, la notación de Chen para ERD (con diamantes para las relaciones) permite relaciones para conectar más de dos tipos de entidad, como se ilustra en la figura 5.15.<sup>2</sup> La relación *Uses* lista a los proveedores y las partes utilizadas en los proyectos. Por ejemplo, una instancia de relaciones que incluya a Supplier1, Part1 y Project1 indica que Supplier1 provee Part1 de Project1. Una relación M-forma que incluye tres tipos de entidad se le conoce como relación ternaria.

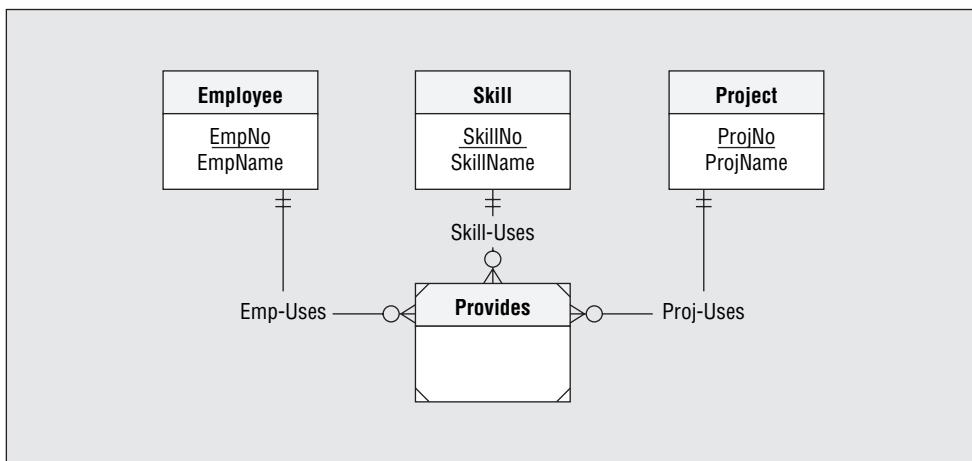
Aunque no puede representar de forma directa las relaciones M-way con la notación de la pata de cuervo, debe comprender cómo representarlas de forma indirecta. Usted usa un tipo de entidad asociativa y una colección de identificadores de relaciones 1-M para representar una relación M-way. En la figura 5.16, tres relaciones 1-M enlazan el tipo de entidad asociativa, *Uses*, con los tipos de entidad *Part*, *Supplier* y *Project*. El tipo de entidad *Uses* es asociativo, ya que su rol es conectar otros tipos de entidad. Como los tipos de entidad asociativos proporcionan un rol de conectividad, algunas veces se les dan nombres de verbos. Además, los tipos de entidad asociativos siempre son débiles ya que piden prestada la llave primaria por completo. Por ejemplo, el tipo de entidad *Uses* obtiene su llave primaria de las tres relaciones que identifica.

Como ejemplo adicional, la figura 5.17 muestra el tipo de entidad asociativa *Provides* que se conecta con los tipos de entidad *Employee*, *Skill* y *Project*. Una instancia de ejemplo del tipo de entidad *Provides* contiene *Employee1* que proporciona *Skill1* en el *Project1*.

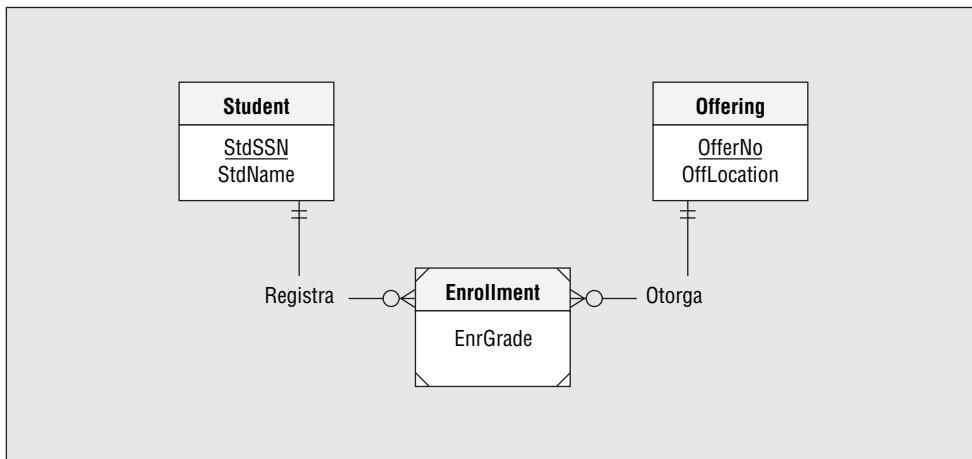
**tipo de entidad asociativa**  
una entidad débil que depende de dos o más tipos de entidades para su llave primaria. Un tipo de entidad asociativa con más de dos relaciones identificables se le conoce como un tipo de entidad asociativa M-way.

<sup>2</sup> La notación de Chen debe su nombre al Dr. Peter Chen, quien publicó el documento que definía el modelo entidad-relación en 1976.

**FIGURA 5.17**  
Tipo de entidad asociativa que conecta  
*Employee*, *Skill* y  
*Project*



**FIGURA 5.18**  
Relación *EnrollsIn*  
M-N (figura 5.10)  
transformada en relaciones 1-M



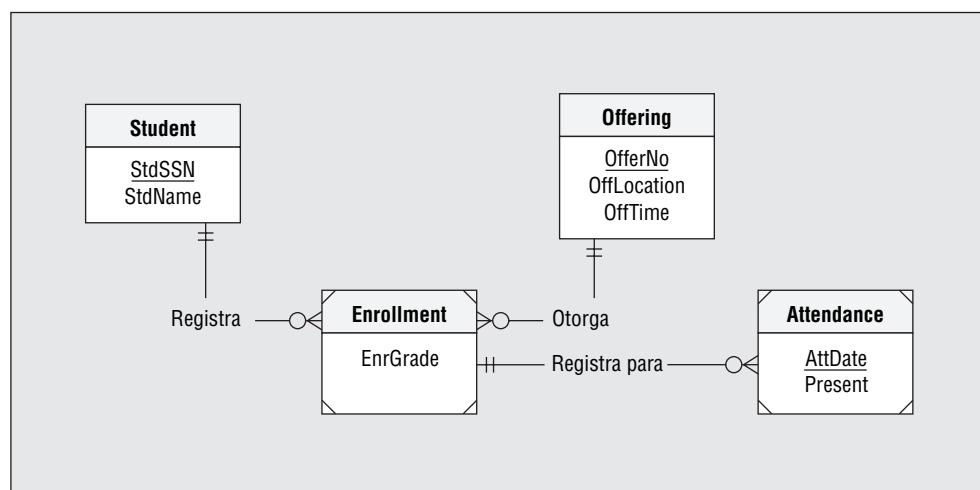
El tema de cuándo utilizar un tipo de entidad asociativa M-way puede ser difícil de comprender (por ejemplo, un tipo de entidad asociativa que represente una relación M-way). Si una base de datos sólo necesita registrar parejas de hechos, no se necesita un tipo de entidad asociativa M-way. Por ejemplo, si una base de datos sólo necesita registrar quién es el proveedor de una parte y qué proyectos usan una parte, entonces no se debe usar un tipo de entidad asociativa M-way. En este caso, deben existir relaciones binarias entre *Supplier* y *Part*, así como entre *Project* y *Part*. Debe usar un tipo de entidad asociativa M-way cuando la base de datos deba registrar combinaciones de tres (o más) entidades en lugar de sólo las combinaciones de dos entidades. Por ejemplo, si una base de datos requiere registrar qué proveedor proporciona las partes de proyectos específicos, se necesita un tipo de entidad asociativa M-way. Debido a la complejidad de las relaciones M-way, el capítulo 7 proporciona una forma para trabajar con ellas utilizando restricciones, mientras que el capítulo 12 proporciona una forma de trabajar con ellas utilizando formularios de captura de datos.

**relación  
de equivalencia**  
una relación M-N puede  
reemplazarse por un tipo  
de entidad asociativa y  
dos relaciones 1-M. En  
la mayoría de los casos  
la elección entre una  
relación M-N y el tipo  
de entidad asociativa  
depende de las preferen-  
cias personales.

### 5.2.3 Equivalencia entre las relaciones 1-M y M-N

Para perfeccionar la comprensión de las relaciones M-N, debe conocer la equivalencia de sus relaciones. Una relación M-N se puede reemplazar por un tipo de entidad asociativa y dos relaciones 1-M. La figura 5.18 muestra la relación *EnrollsIn* (figura 5.10) convertida al estilo 1-M. En la figura 5.18, dos relaciones identificables y un tipo de entidad asociativa reemplazan a la relación *EnrollsIn*. El nombre de la relación (*EnrollsIn*) se cambió por un sustantivo (*Enrollment*) para seguir la convención sobre los sustantivos para los nombres de los tipos de entidad.

**FIGURA 5.19**  
Tipo de entidad de asistencia que se agrega al ERD de la figura 5.18



El estilo 1-M es similar a la representación de un diagrama de bases de datos relacional. Si se siente más cómodo con el estilo 1-M, úselo. En términos del ERD, los estilos M-N y 1-M tienen el mismo significado.

La transformación de una relación M-N en relaciones 1-M es semejante a representar una relación M-way utilizando relaciones 1-M. Cuando se presente una relación M-N como un tipo de entidad asociativa y dos relaciones 1-M, el nuevo tipo de entidad es dependiente de las dos relaciones 1-M, tal como lo muestra la figura 5.18. De forma similar, cuando se representen la relaciones M-way, el tipo de entidad asociativa depende de todas las relaciones 1-M, como se muestra en las figuras 5.16 y 5.17.

Existe una situación en la cual es mejor utilizar el estilo 1-M que el estilo M-N. Debe utilizar el estilo 1-M cuando se deba relacionar una relación M-N con otros tipos de entidad mediante relaciones. Por ejemplo, suponga que además de la inscripción a algún curso que se ofrece se debe registrar la asistencia en cada una de las clases. En esta situación, se prefiere utilizar el estilo 1-M porque es necesario para enlazar una inscripción con los registros de asistencia. La figura 5.19 muestra el tipo de entidad *Attendance* que se agrega al ERD de la figura 5.18. Observe que una relación M-N entre los tipos de entidad *Student* y *Offering* no hubieran permitido otra relación con *Attendance*.

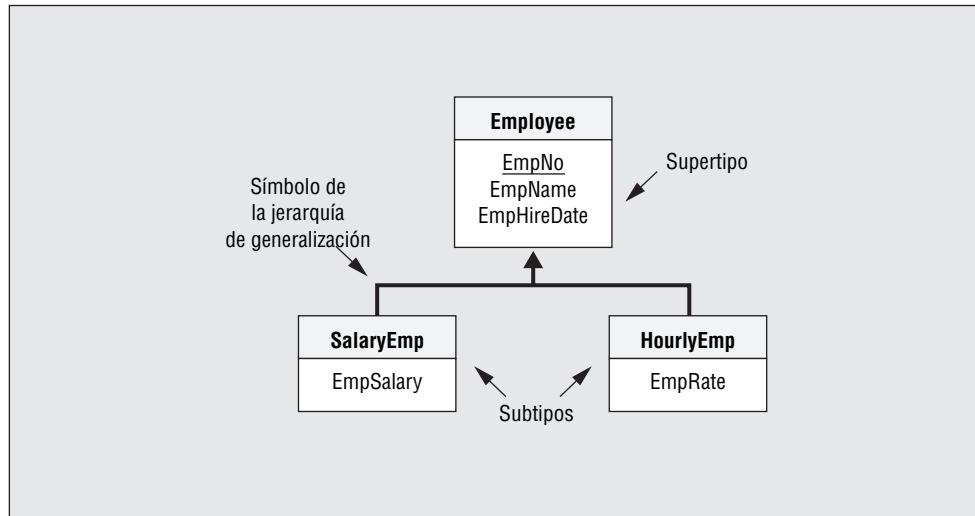
La figura 5.19 proporciona otros ejemplos para identificar dependencias. *Attendance* depende de *Enrollment* en la relación *RecordedFor*. La llave primaria de *Attendance* está formada por *AttDate* junto con la llave primaria de *Enrollment*. De forma similar, *Enrollment* depende de *Student* y *Offering*. La llave primaria de *Enrollment* es la combinación de *StdSSN* y *OfferNo*.

### 5.3 Clasificación en el modelo de entidad-relación

Las personas clasifican a las entidades para mejorar la comprensión de su entorno. Por ejemplo, los animales están clasificados en mamíferos, reptiles y otras categorías para entender las semejanzas y diferencias entre distintas especies. En los negocios, la clasificación también es vital. La clasificación se puede aplicar a las inversiones, empleados, clientes, préstamos, partes, etc. Por ejemplo, cuando se solicita un préstamo hipotecario, hay que hacer una distinción importante entre préstamos hipotecarios fijos y préstamos hipotecarios con tasas variables. Para cada tipo de hipoteca, existen muchas variaciones que se diferencian por características tales como el periodo para volver a pagar, las penalizaciones por prepagos y el monto del préstamo.

Esta sección describe la notación ERD para soportar la clasificación. Aprenderá a usar jerarquías de generalización, a especificar las restricciones de cardinalidad para las jerarquías de generalización, y a usar jerarquías de generalización de varios niveles para las clasificaciones complejas.

**FIGURA 5.20**  
Jerarquía de generalización para empleados



**jerarquía de generalización**  
conjunto de tipos de entidad ordenadas en una estructura jerárquica para mostrar sus atributos similares. Cada subtipo del tipo de entidad hijo contiene un subconjunto de entidades de su supertipo o tipo de entidad padre.

**herencia**  
una característica del modelado de datos que permite compartir los atributos entre un supertipo y un subtipo. El subtipo hereda atributos de su supertipo.

### 5.3.1 Jerarquías de generalización

Las jerarquías de generalización permiten que los tipos de entidad se relacionen por nivel de especialización. La figura 5.20 ilustra una jerarquía de generalización para clasificar a los empleados según su salario contra la clasificación según su horario. Tanto los empleados asalariados como los clasificados por horario son clases especiales de empleados. El tipo de entidad *Employee* se muestra como el supertipo (o padre). Los tipos de entidad *SalaryEmp* y *HourlyEmp* se conocen como subtipos (o hijos). Como cada entidad subtipo es una entidad supertipo, a la relación entre un subtipo y un supertipo se le conoce como ISA. Por ejemplo, un empleado asalariado es un empleado. Ya que el nombre de la relación (ISA) siempre es el mismo, no se muestra en el diagrama.

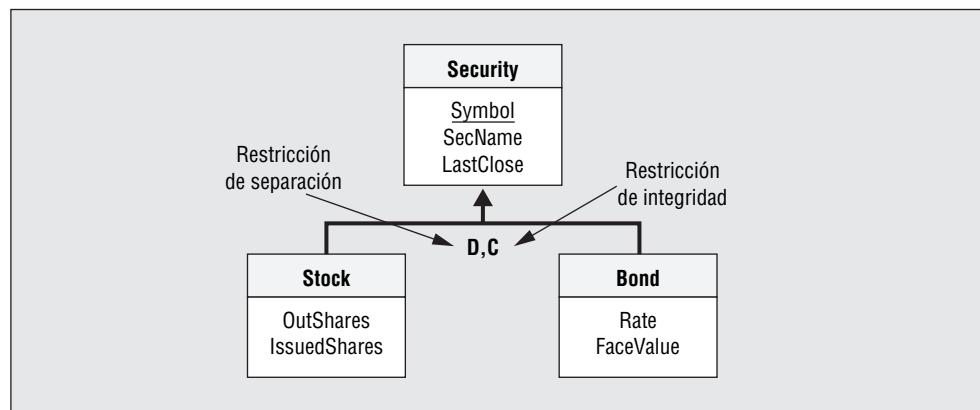
La herencia respalda la característica de compartir elementos entre un supertipo y sus subtipos. Debido a que cada entidad subtipo también es una entidad supertipo, los atributos de supertipo también se aplican a todos los subtipos. Por ejemplo, cada entidad de *SalaryEmp* tiene un número de empleado, nombre y fecha de contratación porque también es una entidad de *Employee*. Herencia significa que los atributos de un supertipo automáticamente forman parte de sus subtipos. Esto es, cada subtipo hereda los atributos de su supertipo. Por ejemplo, los atributos del tipo de entidad *SalaryEmp* son sus atributos directos (*EmpSalary*) y sus atributos heredados de *Employee* (*EmpNo*, *EmpName*, *EmpHireDate*, etc.). Los atributos heredados no se muestran en un ERD. Cuando tenga un subtipo, asuma que éste hereda los atributos de su supertipo.

### 5.3.2 Restricciones de separación e integridad

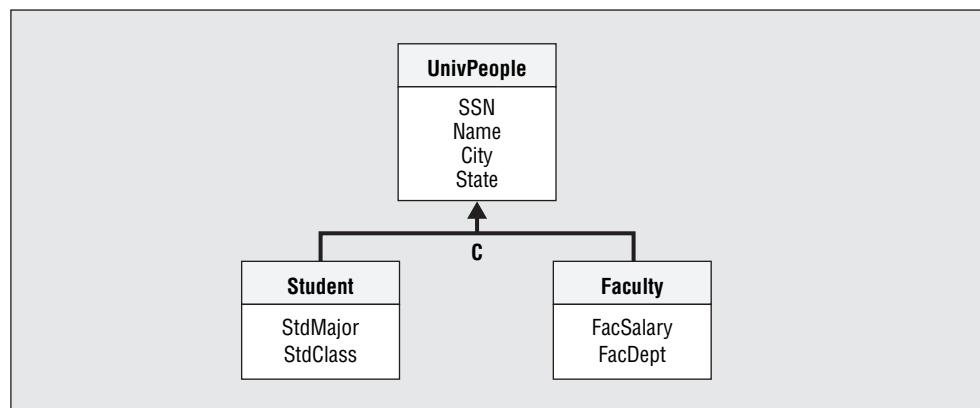
Las jerarquías de generalización no muestran las cardinalidades debido a que siempre son las mismas. En su lugar, se muestran las restricciones de separación e integridad. Separación significa que los subtipos en una jerarquía de generalización no tienen ninguna entidad en común. En la figura 5.21, la jerarquía de generalización está separada, ya que un título no puede ser un bono y una acción al mismo tiempo. En contraste, la jerarquía de generalización de la figura 5.22 no está separada porque los asistentes que enseñan pueden ser estudiantes y catedráticos. Por ende, el conjunto de estudiantes sobrepasa al conjunto de catedráticos. Integridad significa que cada entidad de un supertipo debe ser una entidad en uno de los subtipos en la jerarquía de generalización. La restricción de integridad de la figura 5.21 significa que cada título debe ser o una acción o un bono.

Algunas jerarquías de generalización carecen de las restricciones de separación e integridad. En la figura 5.20, la carencia de la restricción de la separación significa que algunos empleados pueden recibir pago en salario y en horas. La carencia de la restricción de integridad indica que a algunos empleados no se les paga por salario o por hora (tal vez por comisión).

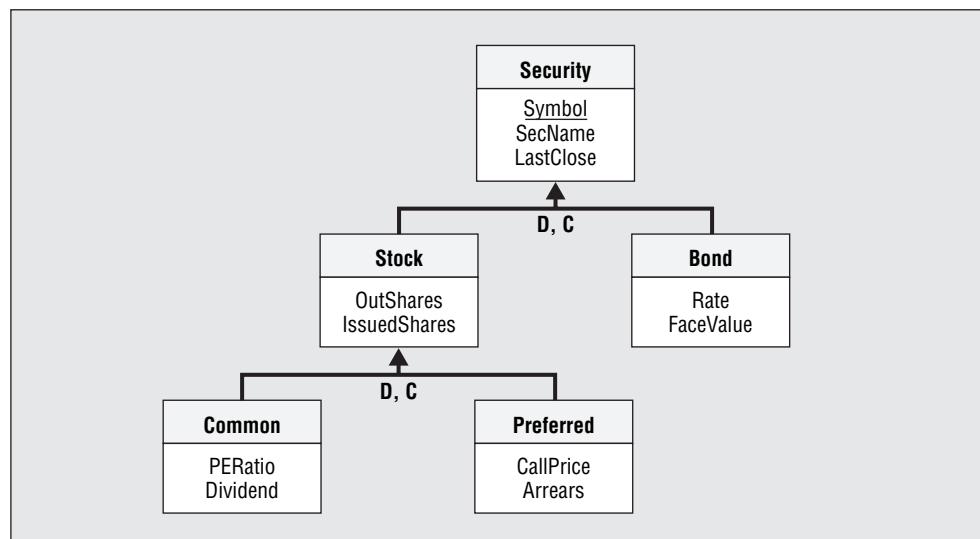
**FIGURA 5.21**  
Jerarquía de generalización para títulos



**FIGURA 5.22**  
Jerarquía de generalización para personas de la universidad



**FIGURA 5.23**  
Niveles múltiples de jerarquías de generalización



### 5.3.3 Niveles múltiples de generalización

Las jerarquías de generalización pueden ampliarse a más de un nivel. Esta práctica puede ser útil en disciplinas como las inversiones, donde el conocimiento es altamente estructurado. En la figura 5.23, existen dos niveles o subtipos entre los títulos. La herencia se amplia a todos los

subtipos, directos e indirectos. Por ende, los tipos de entidad *Common* y *Preferred* heredan los atributos de *Stock* (el padre inmediato) y *Security* (el padre indirecto). Observe que las restricciones de la separación e integridad se pueden hacer para cada grupo de subtipos.

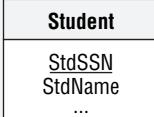
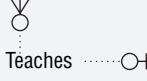
## 5.4 Resumen de notación y reglas de diagramación

En las secciones previas de este capítulo se ha hablado mucho sobre la notación ERD. En esta sección se proporciona un resumen adecuado y las reglas que ayudarán a evitar los errores más comunes de la diagramación.

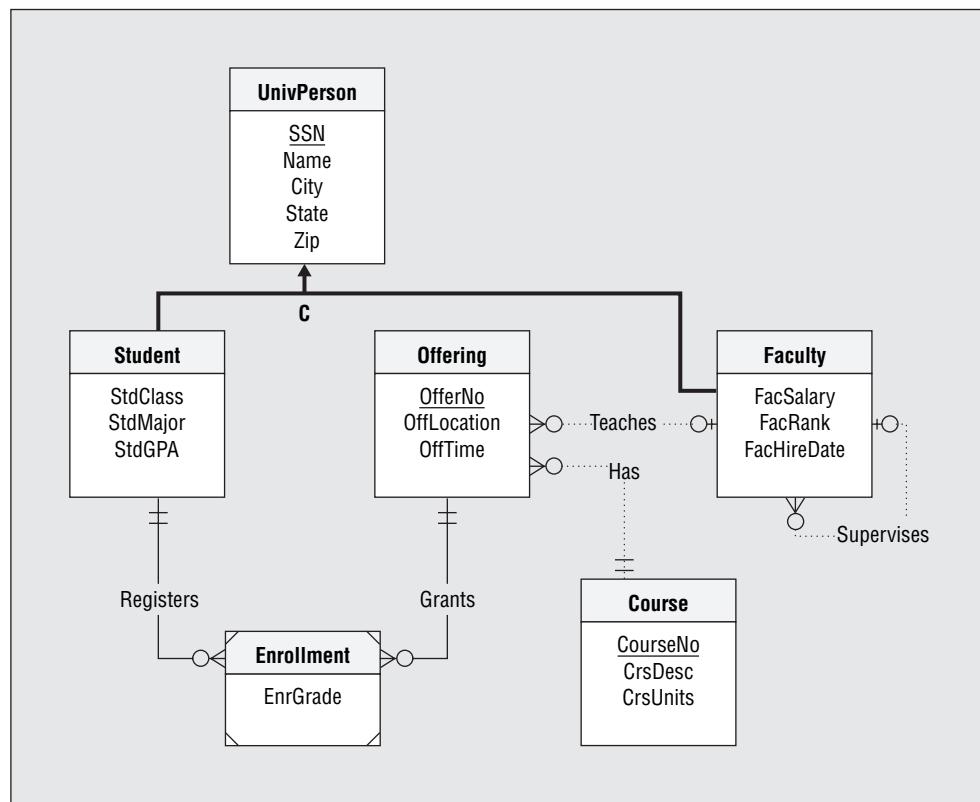
### 5.4.1 Resumen de notación

La tabla 5.2 presenta un resumen para ayudarle a recordar la notación introducida en las secciones previas, mientras que la figura 5.24 aplica la notación a la base de datos de la universidad del capítulo 4. La figura 5.24 difiere en algunas formas de la base de datos de la universidad del capítulo 4 para ilustrar gran parte de la notación de pata de cuervo. La figura 5.24 contiene una jerarquía de generalización para ilustrar la semejanza entre los estudiantes y catedráticos. Debe observar que la llave primaria de los tipos de entidad *Student* y *Faculty* es *SSN*, un atributo heredado del tipo de entidad *UnivPerson*. El tipo de entidad *Enrollment* (asociativo) y las relaciones identificables (*Registers* y *Grants*) pueden aparecer como una relación M-N tal como se mostró previamente en la figura 5.10. Además de estos elementos, la figura 5.24 omite algunos atributos con fines de simplicidad.

**TABLA 5.2** Resumen de la notación de pata de cuervo

Símbolo	Significado
	Tipo de entidad con atributos (llave primaria subrayada).
	Relación M-N con atributos: los atributos se muestran si se permite la habitación. De lo contrario los atributos se listan de forma separada.
	Identificación de las dependencias: identificación de relaciones (líneas sólidas de relaciones) y entidades débiles (líneas diagonales en las esquinas del rectángulo). Los tipos de entidad asociativa también son débiles, ya que son dependientes (por definición).
	Jerarquía de generalización con restricciones de separación e integridad.
	Cardinalidad de la dependencia existente (cardinalidad mínima de 1): el símbolo interno es una línea perpendicular a la línea de relación.
	Cardinalidad opcional (cardinalidad mínima de 0): el símbolo interno es un círculo.
	Cardinalidad de un sólo valor (cardinalidad máxima de 1): el símbolo externo es una línea perpendicular.

**FIGURA 5.24**  
ERD para la base de datos de la universidad



### Representación de reglas de negocio en un ERD

Conforme desarrolle un ERD, recordará que contiene reglas de negocio que obligan a que se lleven a cabo las políticas de la organización y se promueva una comunicación eficiente entre los involucrados en el negocio. Un ERD contiene importantes reglas de negocio representadas como llaves primarias, relaciones, cardinalidades y jerarquías de generalización. Las llaves primarias respaldan la identificación de entidades, un requerimiento importante en la comunicación de negocios. La identificación de dependencias incluye una entidad que depende de otras entidades para su identificación, un requerimiento en ciertas comunicaciones de negocio. Las relaciones indican las conexiones directas entre las unidades de la comunicación de negocio. Las cardinalidades restringen el número de entidades relacionadas en las relaciones que respaldan las políticas de la organización y la comunicación constante del negocio. Las jerarquías de generalización con restricciones de separación e integridad respaldan la clasificación de las entidades de negocio y políticas organizacionales. Por ende, los elementos de un ERD son cruciales para obligar a que se lleven a cabo las políticas organizacionales y una eficiente comunicación de negocio.

Para otros tipos de restricciones de negocios, un ERD se puede mejorar con documentación informal o con un lenguaje de reglas formales. Debido a que SQL:2003 soporta un lenguaje de reglas formales (véanse los capítulos 11 y 14), no se propone aquí algún lenguaje. En la ausencia de un lenguaje de reglas formales, las reglas de negocio se pueden almacenar como documentación informal asociada con tipos de entidad, atributos y relaciones. Ejemplos típicos para especificar las reglas de negocio como documentación informal son las restricciones de llaves candidatas, restricciones de comparación de atributos, restricciones de valores nulos y valores por omisión. Las llaves candidatas proporcionan formas alternativas para identificar las entidades de negocios. Las restricciones para comparar atributos restringen los valores de los atributos a un conjunto fijo de valores o a los valores de otros atributos. Las restricciones de valores nulos y de valores por omisión respaldan las políticas de integridad de las actividades de colección de datos.

**TABLA 5.3**  
**Resumen de las reglas de negocio en un ERD**

Regla de negocio	Representación ERD
Identificación de entidades	Llaves primarias para los tipos de entidad, identificación de dependencias (entidades débiles y relaciones identificables), documentación informal acerca de otros atributos únicos
Conexión entre entidades de negocio	Relaciones
Número de entidades relacionadas	Cardinalidades mínimas y máximas
Inclusión entre conjuntos de entidades	Jerarquías de generalización
Valores razonables	Documentación informal de restricción de atributos (comparación de valores constantes con otros atributos)
Integridad de la colección de datos	Documentación informal de valores nulos y valores por omisión

**TABLA 5.4**  
**Reglas de consistencia e integridad**

Tipo de regla	Descripción
<i>Integridad</i>	<ol style="list-style-type: none"> <li><b>Regla de la llave primaria:</b> todos los tipos de entidad tienen una llave primaria (directa, prestada o heredada).</li> <li><b>Regla de denominación:</b> todos los tipos de entidad, relaciones y atributos tienen nombre.</li> <li><b>Regla de cardinalidad:</b> la cardinalidad está dada para los dos tipos de entidad de una relación.</li> <li><b>Regla de participación de la entidad:</b> todos los tipos de entidad participan en al menos una relación, excepto los de una jerarquía de generalización.</li> <li><b>Regla de participación en una jerarquía de generalización:</b> cada jerarquía de generalización participa en al menos una relación con un tipo de entidad que no está en la jerarquía de generalización.</li> </ol>
<i>Consistencia</i>	<ol style="list-style-type: none"> <li><b>Regla de nombres de entidad:</b> los nombres de tipo de entidad son únicos.</li> <li><b>Regla de nombre de atributo:</b> los nombres de atributos son únicos dentro de los tipos de entidad y relaciones.</li> <li><b>Regla de nombre de atributos heredados:</b> los nombres de los atributos de un subtipo no coinciden con los nombres de los atributos heredados (directos o indirectos).</li> <li><b>Regla de tipo de conexión relación/entidad:</b> todas las relaciones conectan dos tipos de entidad (no necesariamente distintos).</li> <li><b>Regla de conexión relación/relación:</b> las relaciones no se conectan con otras relaciones.</li> <li><b>Regla de entidad débil:</b> las entidades débiles tienen al menos una relación identificable.</li> <li><b>Regla de relación identifiable:</b> para cada relación identifiable, al menos uno de los tipos de entidad participante debe ser débil.</li> <li><b>Regla de cardinalidad de la identificación de dependencias:</b> para cada relación identifiable, la cardinalidad mínima y máxima debe ser 1 en el sentido del tipo de entidad hijo (entidad débil) al tipo de entidad padre.</li> <li><b>Regla de la llave foránea redundante:</b> las llaves foráneas redundantes no se usan.</li> </ol>

La tabla 5.3 resume las categorías comunes de las reglas de negocio que se pueden especificar de manera formal o informal en un ERD.

#### 5.4.2 Reglas de diagramación

Para proporcionar una guía sobre el uso apropiado de la notación, la tabla 5.4 presenta las reglas de integridad y consistencia con el fin de ofrecerle una guía sobre el uso correcto de la notación. Debe aplicar estas reglas cuando complete un ERD para asegurarse de que no existen errores en la notación de su ERD. Por ende, las reglas de diagramación tienen un propósito similar a las reglas de sintaxis de un lenguaje de programación. La ausencia de errores de sintaxis no significa que un programa de cómputo realice sus tareas correctamente. Por lo mismo, la ausencia de

errores en la notación no significa que un ERD proporcione una representación adecuada de los datos. Las reglas de diagramación no aseguran que usted haya considerado varias alternativas, representando los requerimientos del usuario de forma adecuada y documentado su diseño. El capítulo 6 describe estos elementos para mejorar sus habilidades en el modelado de datos.

La mayoría de las reglas de la tabla 5.4 no requieren de mucha elaboración. Las primeras tres reglas de integridad y las primeras cinco reglas de consistencia son fáciles de entender. Aunque las reglas son simples, debe revisar que sus ERD las cumplan ya que fácilmente puede pasar por alto alguna violación en un ERD de tamaño moderado.

Las reglas de consistencia no requieren de nombres únicos para las relaciones ya que los tipos de entidades participantes proporcionan un contexto para los nombres de las relaciones. Sin embargo, es buena práctica utilizar nombres únicos para las relaciones tanto como sea posible con el fin de poder distinguir entre relaciones. Como no es normal tener más de una relación entre los mismos tipos de entidades, las reglas de consistencia no incluyen esta previsión.

Las reglas de integridad 4 (regla de participación de la entidad) y 5 (regla de participación en una jerarquía de generalización) requieren elaboración. La violación de estas reglas es una advertencia, no necesariamente un error. En la mayoría de los ERDs, todos los tipos de entidad que no estén en una jerarquía generalizada y todas las jerarquías generalizadas están conectadas al menos con otro tipo de entidad. En raras situaciones un ERD contiene un tipo de entidad desconectado que sirve sólo para almacenar una lista de entidades. La regla 5 se aplica a toda la jerarquía de generalización, no a cada uno de los tipos de entidad en una jerarquía de generalización. En otras palabras, al menos un tipo de entidad en una jerarquía de generalización debe estar conectada con al menos un tipo de entidad que no esté en la jerarquía de generalización. En muchas jerarquías de generalización, los tipos de entidad múltiple participan en las relaciones. Las jerarquías de generalización permiten que los subtipos participen en las relaciones, por ende la participación de las relaciones de restricción. Por ejemplo, en la figura 5.24, *Student* y *Faculty* participan en las relaciones.

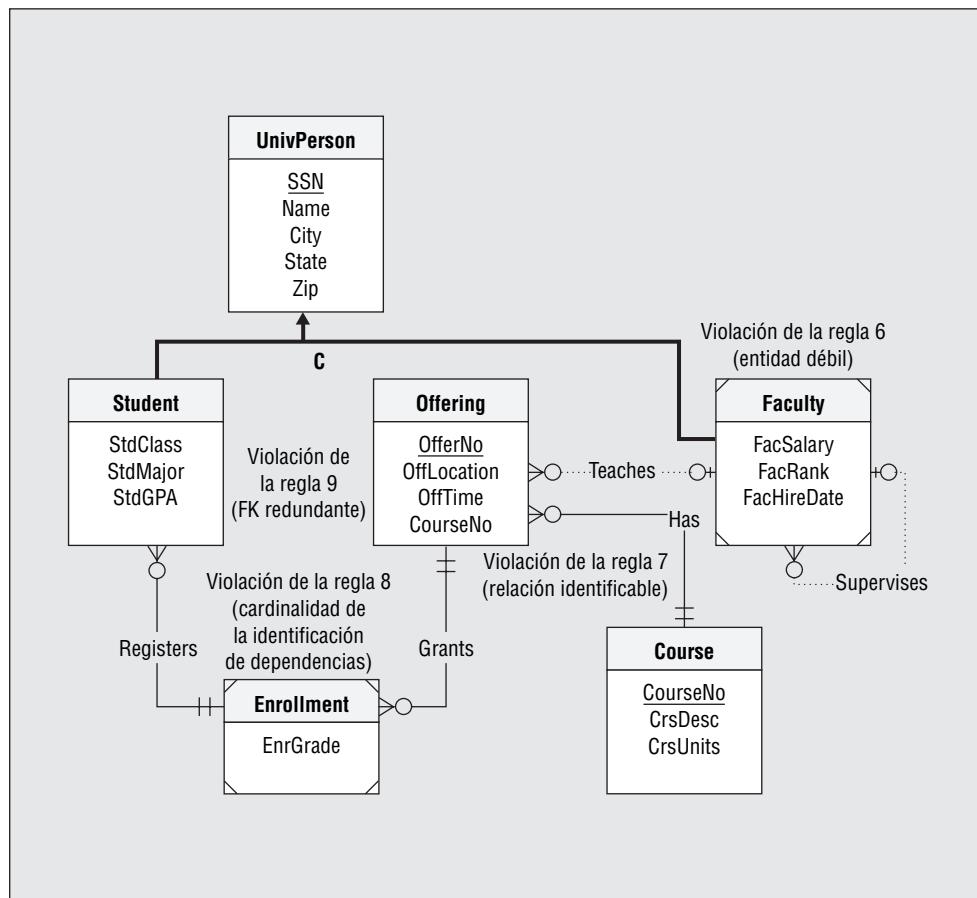
Las reglas de consistencia 6 a 9 incluyen errores comunes de los ERDs por parte de modeladores de datos novatos. Los modeladores de datos novatos violan las reglas de consistencia 6 a 8 debido a la complejidad de la identificación de dependencias. La identificación de dependencias involucra a una entidad débil y las relaciones identificables proporcionan más fuente de errores que otras partes de la notación de pata de cuervo. Además, cada relación identificable también requiere una cardinalidad mínima y máxima de 1 en dirección del tipo de entidad hijo (entidad débil) al padre. Los modeladores de datos novatos violan la regla de consistencia 9 (regla de la llave foránea redundante) debido a la confusión entre un ERD y el modelo de datos relacional. El proceso de conversión transforma las relaciones 1-M en llaves foráneas.

### *Ejemplo de violaciones de reglas y resoluciones*

Debido a que las reglas de identificación de dependencias y la regla de llave foránea redundante son una fuente de errores para los diseñadores novatos, esta sección proporciona un ejemplo para ilustrar violaciones y resoluciones a las reglas. La figura 5.25 demuestra violaciones a las reglas de identificación de dependencias (reglas de consistencia 6 a 9) y la regla de llave foránea redundante (regla de consistencia 9) para el ERD de la base de datos de la universidad. La siguiente lista explica las violaciones:

- **Violación a la regla de consistencia 6 (regla de la entidad débil):** *Faculty* no puede ser una entidad débil sin al menos una relación identificable.
- **Violación a la regla de consistencia 7 (regla de identificación de relaciones):** La relación *Has* es una relación identificable, pero ni *Offering* ni *Course* son entidades débiles.
- **Violación a la regla de consistencia 8 (regla de cardinalidad de identificación de dependencias):** La cardinalidad de la relación *Registers* de *Enrollment* a *Student* debe ser (1,1) y no (0, Muchos).
- **Violación a la regla de consistencia 9 (regla de la llave foránea redundante):** El atributo *CourseNo* en el tipo de entidad *Offering* es redundante con la relación *Has*. Debido a que *CourseNo* es la llave primaria de *Course*, no debe ser atributo de *Offering* el que enlace *Offering* con *Course*. La relación *Has* proporciona un enlace hacia *Course*.

**FIGURA 5.25**  
ERD con violaciones a las reglas de consistencia 6 a 9



Para la mayoría de las reglas la solución de violaciones resulta sencilla. La tarea principal es reconocer la violación. Para identificar las reglas de dependencia, la solución puede depender de los detalles del problema. La siguiente lista sugiere acciones correctivas que pueden efectuarse sobre los errores de los diagramas:

- Solución a la regla de consistencia 6 (regla de la entidad débil):** El problema puede resolverse agregando una o más relaciones identificables o modificando la entidad débil en una entidad normal. En la figura 5.25, el problema se resuelve al hacer que **Faculty** sea una entidad normal. La solución más común es agregar una o más relaciones identificables.
- Solución a la regla de consistencia 7 (regla de identificación de relaciones):** El problema se puede resolver agregando una entidad débil o haciendo que la relación sea no identificable. En la figura 5.25, el problema se resuelve al hacer que la relación **Has** no sea identifiable. Si existe más de una relación identifiable que incluya al mismo tipo de entidad, la solución típica incluye la designación del tipo de entidad común como una entidad débil.
- Solución a la regla de consistencia 8 (regla de cardinalidad de identificación de dependencias):** El problema se puede resolver al modificar la cardinalidad de la entidad débil a (1,1). Por lo general se invierte la cardinalidad de la relación identifiable. En la figura 5.25, la cardinalidad de la relación **Registers** debe invertirse (1,1), cerca de **Student** y (0,Muchos) cerca de **Enrollment**.
- Solución a la regla de consistencia 9 (regla de la llave foránea redundante):** Normalmente el problema se puede resolver eliminando la llave foránea redundante. En la figura 5.25, **CourseNo** debe eliminarse como atributo de **Offering**. En algunos casos, el atributo no debe representar una llave foránea. Si el atributo no presenta una llave foránea, debe renombrarse en lugar de eliminarse.

**TABLA 5.5**  
**Organización de reglas alternativas**

Categoría	Reglas
<i>Nombres</i>	<p>Se nombran todos los tipos de entidades, relaciones y atributos. (Regla de integridad 2)</p> <p>Los nombres de los tipos de entidad son únicos (Regla de consistencia 1)</p> <p>Los nombres de los atributos son únicos para cada tipo de entidad y relación. (Regla de consistencia 2)</p>
<i>Contenido</i>	<p>Los nombres de los atributos de un subtipo no coinciden con los nombres de los atributos heredados (directos o indirectos). (Regla de consistencia 3)</p> <p>Todos los tipos de entidad tienen una llave primaria (directa, prestada o heredada). (Regla de integridad 1)</p>
<i>Conexión</i>	<p>La cardinalidad está dada por las dos entidades que participan en una relación. (Regla de integridad 3)</p> <p>Todos los tipos de entidad participan en al menos una relación, excepto aquellos en una jerarquía de generalización. (Regla de integridad 4)</p> <p>Cada jerarquía de generalización participa en al menos una relación con un tipo de entidad, no en la jerarquía de generalización. (Regla de integridad 5)</p> <p>Todas las relaciones conectan dos tipos de entidad. (Regla de consistencia 4)</p> <p>Las relaciones no se conectan con otras relaciones. (Regla de consistencia 5)</p>
<i>Identificación de dependencias</i>	<p>No se usan las llaves foráneas redundantes. (Regla de consistencia 9)</p> <p>Las entidades débiles tienen al menos una relación identificable. (Regla de consistencia 6)</p> <p>Para cada relación identificable, al menos una de las entidades participantes debe ser débil. (Regla de consistencia 7)</p> <p>Para cada entidad débil, la cardinalidad mínima y máxima debe ser igual a 1 para cada relación identificable. (Regla de consistencia 8)</p>

### *Organización alternativa de las reglas*

La organización de las reglas de la tabla 5.4, puede ser difícil de recordar. La tabla 5.5 proporciona una agrupación alternativa para este propósito. Si encuentra que esta organización es más intuitiva, debe utilizarla. Sin embargo, si decide recordar las reglas, el punto importante es aplicarlas después de haber completado el ERD. Para ayudarle a aplicar las reglas de diagramación, la mayoría de las herramientas CASE realizan revisiones específicas de las notaciones soportadas por las herramientas. La siguiente sección describe la revisión de las reglas de diagramación por medio del ER Assistant, la herramienta de modelado de datos disponible con este libro de texto.

### *Soporte para el ER Assistant*

Para mejorar la productividad de los modeladores de datos novatos, el ER Assistant soporta las reglas de consistencia 4 y 5 mediante sus herramientas de diagramación. Las relaciones deben estar conectadas con dos tipos de entidades (no necesariamente distintas) prohibiendo la violación a las reglas de consistencia 4 y 5. Para las otras reglas de integridad y consistencia, el ER Assistant proporciona el botón Check Diagram que genera un reporte de violación a las reglas. Debido a que el botón Check Diagram puede utilizarse cuando aún no está completo el ERD, el ER Assistant no necesita reparar las violaciones a las reglas encontradas en un ERD. Antes de completar un ERD, debe atender cada una de las violaciones identificada por la herramienta ER Assistant.

Para la regla de llave foránea redundante (regla de consistencia 9), el ER Assistant usa una implementación sencilla para determinar si un ERD contiene una llave foránea redundante. El ER Assistant revisa el tipo de entidad hijo (tipo de entidad en el lado “muchos” de la relación) para buscar un atributo con el mismo nombre y tipo de datos que la llave primaria del tipo de entidad padre (tipo de entidad en el lado “uno” de la relación). Si el ER Assistant encuentra un atributo con el mismo nombre y tipo de dato, se lista una violación en el reporte Revisión del Diagrama.

## 5.5 Comparación a otras notaciones

La notación ERD presentada en este capítulo es semejante pero no idéntica a lo que encontrará más adelante. No existe un estándar para la notación de los ERDs. Tal vez existen seis notaciones de ERD bastante populares, cada una con sus pequeñas variaciones que aparecen en la práctica. La notación en este capítulo proviene de la plantilla de pata de cuervo incluida en Visio Professional 5 con el agregado para la notación de generalizaciones. Las notaciones que encontrará en la práctica dependerán de factores tales como la herramienta del modelado de datos (si existe) que use su organización y la industria. Una cosa es segura: debe estar preparado para adaptarse a la notación utilizada. Esta sección describe variaciones que puede encontrar en los ERD, así como la notación de diagramas de clases del Lenguaje de Modelado Unificado (UML), un estándar emergente para el modelado de datos.

### 5.5.1 Variaciones ERD

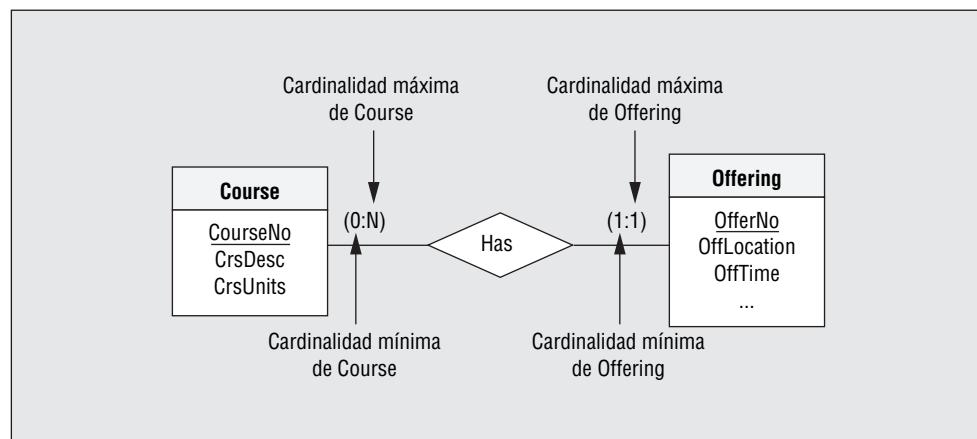
Como no existe una amplia aceptación de un estándar de ERD, se pueden utilizar distintos símbolos para representar el mismo concepto. Las cardinalidades de las relaciones son una fuente de grandes variaciones. Usted debe poner atención en la colocación de los símbolos de cardinalidad. La notación en este capítulo coloca los símbolos cerca del tipo de entidad “lejano”, mientras que otras notaciones colocan los símbolos de cardinalidad cerca del tipo de entidad “cercano”. La notación de este capítulo usa una representación visual de las cardinalidades con las cardinalidades mínimas y máximas identificadas por tres símbolos. Otras notaciones usan una representación textual con letras y enteros en lugar de símbolos. Por ejemplo, la figura 5.26 muestra un ERD con la notación de Chen con la posición de las cardinalidades invertida, cardinalidades ilustradas con texto y cardinalidades representadas con diamantes.

Otras variaciones a los símbolos son representaciones visuales para cierta categoría de tipos de entidad. En algunas notaciones, las entidades débiles y las relaciones M-N tienen representaciones especiales. Las entidades débiles algunas veces se encuentran encerradas entre rectángulos dobles. Las relaciones identificables algunas veces están encerradas con diamantes dobles. Las relaciones M-N con atributos algunas veces se muestran como un rectángulo con un diamante adentro que representa la calidad dual (relación y tipo de entidad).

Además de las variaciones a los símbolos, existen también variaciones a las reglas, tal como se muestra en la siguiente lista. Para cada restricción existe un remedio. Por ejemplo, si sólo se soportan relaciones binarias, las relaciones M-way deben representarse como un tipo de entidad asociativa con relaciones 1-M.

1. Algunas notaciones no soportan las relaciones M-way.
2. Algunas notaciones no soportan las relaciones M-N.
3. Algunas notaciones no soportan las relaciones con atributos.

**FIGURA 5.26**  
Notación de Chen  
para el ERD Course-  
Offering



4. Algunas notaciones no soportan las relaciones autorreferenciadas (unitarias).
5. Algunas notaciones permiten que las relaciones se conecten con otras relaciones.
6. Algunas notaciones muestran las llaves foráneas como atributos.
7. Algunas notaciones permiten que los atributos tengan más de un valor (atributos multivaleores).

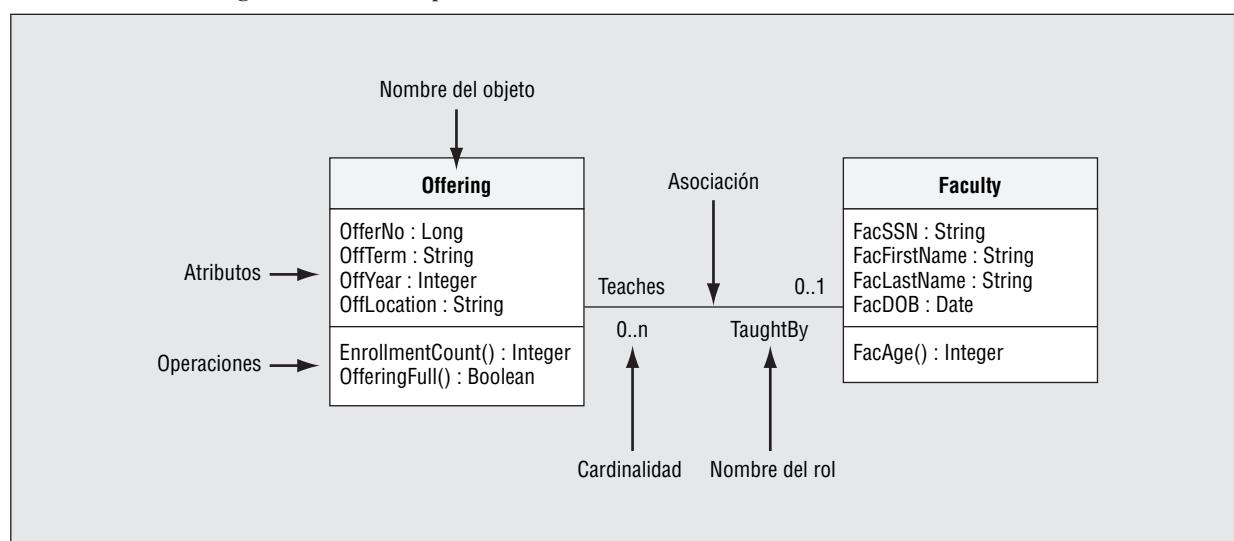
Las restricciones en la notación de un ERD no necesariamente hacen que la notación sea menos expresiva que otras notaciones sin restricciones. Pueden ser necesarios símbolos adicionales en el diagrama, pero aún así se pueden representar los mismos conceptos. Por ejemplo, la notación de pata de cuervo no soporta las relaciones M-way. Sin embargo, las relaciones M-way pueden representarse con el uso de tipos de entidad asociativa M-way. Los tipos de entidad asociativa M-way requieren de símbolos adicionales que no requieren las relaciones M-way, pero se representan los mismo conceptos.

### 5.5.2 Notación del diagrama de clases del lenguaje de modelado unificado

El lenguaje de modelado unificado se ha convertido en la notación estándar del modelado orientado a objetos. El modelado orientado a objetos se enfocan sobre objetos en lugar de procesos, como se enfatiza en las metodologías tradicionales de los desarrollos de sistemas. En el modelado orientado a objetos, primero se definen los objetos, seguidos por sus características (atributos y operaciones) y después la integración dinámica entre ellos. El UML contiene diagramas de clases, diagramas de interfaces y diagramas de interacción para respaldar el modelado orientado a objetos. La notación del diagrama de clases proporciona una alternativa a las notaciones de los ERD presentados en este capítulo.

Los diagramas de clases contienen clases (colección de objetos), asociaciones entre las clases (relaciones binarias) y características de los objetos (atributos y operaciones). La figura 5.27 muestra un diagrama de clases sencillo con las clases *Offering* y *Faculty*. El diagrama se dibujó con la plantilla UML de Visio Professional. La asociación de la figura 5.27 representa una relación 1-M. UML soporta los nombres de roles y cardinalidades (mínimas y máximas) para cada dirección de una asociación. La cardinalidad 0..1 significa que un objeto *Offering* se puede relacionar con un mínimo de cero objetos *Faculty* y un máximo de un objeto *Faculty*. Las operaciones se listan debajo de los atributos. Cada operación contiene una lista entre paréntesis de los parámetros, junto con el tipo de datos que regresa la operación.

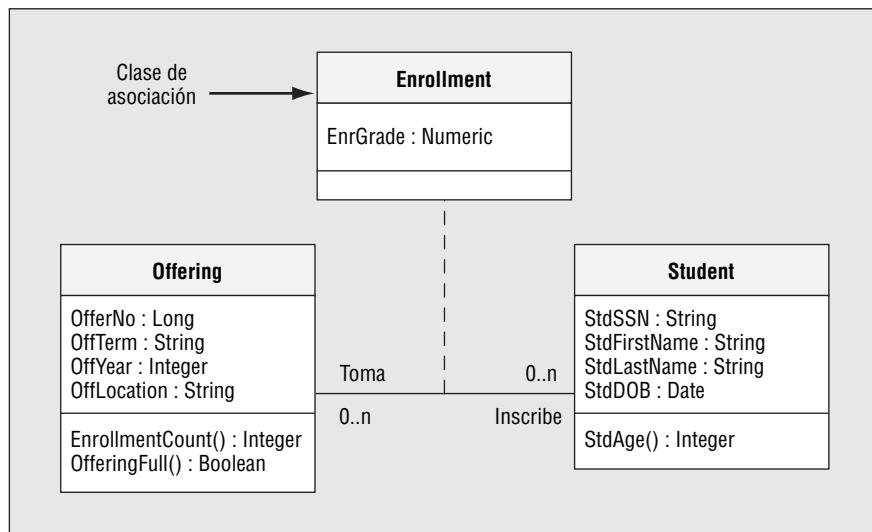
**FIGURA 5.27** Diagrama de clase simple



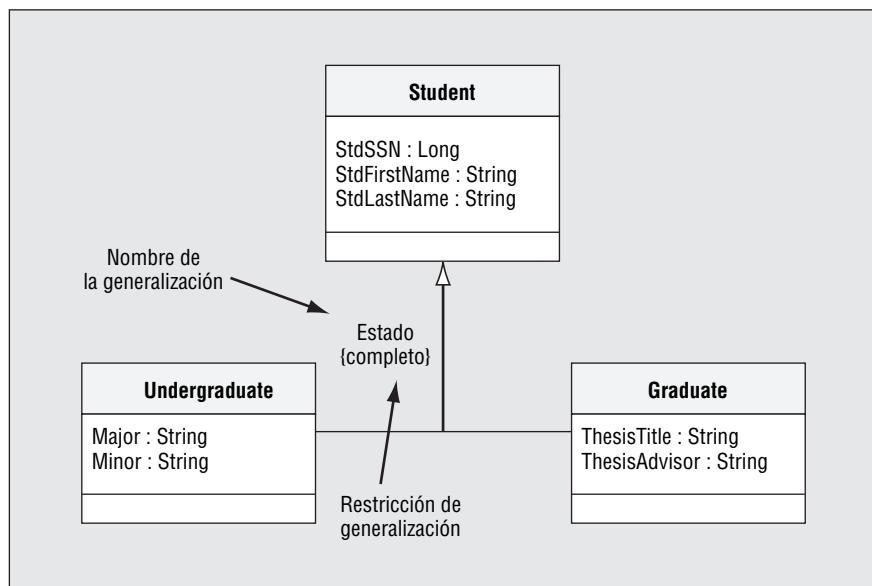
Las asociaciones en UML son semejantes a las relaciones en la notación de pata de cuervo. Las asociaciones pueden representar relaciones binarias o unitarias. Para representar relaciones M-way se requiere una clase y una colección de asociaciones. Para representar una relación M-N con atributos, UML proporciona la clase de asociación para permitir asociaciones que tengan atributos y operaciones. La figura 5.28 muestra una clase de asociación que representa una relación M-N entre las clases *Student* y *Offering*. La clase de asociación contiene los atributos de la relación.

A diferencia de la mayoría de las notaciones de los ERD, el soporte para la generalización está construido dentro de UML desde el principio. En la mayoría de las notaciones de los ERD, se agregó la generalización como una característica adicional después de haber establecido la notación. En la figura 5.29, la flecha grande vacía representa una clasificación de la clase *Student* con las clases *UnderGraduate* y *Graduate*. UML soporta los nombres de generalizaciones y restricciones. En la figura 5.29, la generalización de *Status* está completa, lo que significa que cada estudiante debe ser un estudiante graduado o no.

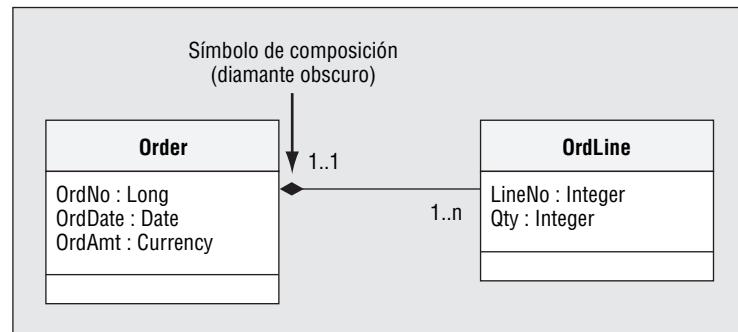
**FIGURA 5.28**  
Clase de asociación que representa una relación M-N con atributos



**FIGURA 5.29**  
Diagrama de clases con una relación generalizada



**FIGURA 5.30**  
**Diagrama de clases con una relación compuesta**



El UML también proporciona un símbolo especial para las relaciones compuestas, semejante a la identificación de dependencias de las notaciones ERD. En una relación compuesta, los objetos de una clase hijo pertenecen sólo a los objetos de la clase padre. En la figura 5.30, cada objeto *OrdLine* pertenece a un objeto *Order*. La eliminación de un objeto padre ocasiona la eliminación del objeto hijo relacionado. Como consecuencia, el objeto hijo generalmente pide prestada parte de la llave primaria del objeto padre. Sin embargo, UML no requiere de la identificación de esta dependencia.

Los diagramas de clases de UML proporcionan muchas otras características que no se presentan en este breve bosquejo. El UML soporta distintos tipos de clases para integrar las preocupaciones de los lenguajes de programación con las preocupaciones del modelado de datos. Otros tipos de clases incluyen clases de valor, clases estereotipo, clases parametrizadas y clases abstractas. Para la generalización, el UML soporta restricciones adicionales tales como la clasificación estática y dinámica y diferentes interpretaciones de relaciones de generalización (subtipo y subclases). Para la integridad de datos, el UML soporta la especificación de restricciones en un diagrama de clases.

Debe observar que los diagramas de clases sólo son una parte del UML. Para ampliar más los conceptos se deben comprender los diagramas de clases en el contexto del modelado orientado a objetos y del UML en su totalidad. Debe esperar dedicarse al término completamente académico para comprender el modelado orientado a objetos y el UML.

## Reflexión final

Este capítulo explicó la notación de diagramas de entidad-relación como prerequisito para aplicar los diagramas de entidad-relación en el proceso de desarrollo de bases de datos. Este capítulo describió los símbolos, importantes patrones de relaciones y jerarquías de generalización para el uso de la notación de pata de cuervo. Los símbolos básicos son tipos de entidad, relaciones, atributos y cardinalidades para ilustrar el número de entidades que participan en una relación. Se describieron cuatro patrones de relación importantes: relaciones muchos-a-muchos (M-N) con atributos, tipos de entidad asociativos que representan las relaciones M-way, relaciones identificables que proporcionan las llaves primarias de las entidades débiles, y relaciones auto-referenciadas (unitarias). Las jerarquías de generalización permiten la clasificación de los tipos de entidad para ilustrar las semejanzas entre los tipos de entidad.

Para mejorar el uso de la notación de pata de cuervo, se presentaron las representaciones de las reglas de negocios, reglas de los diagramas y comparaciones con otras notaciones. Este capítulo presentó representaciones formales e informales de las reglas de negocios de un diagrama de entidad-relación para proporcionar un contexto organizacional para los diagramas de entidad-relación. Las reglas de los diagramas incluyen los requerimientos de integridad y consistencia. Las reglas de los diagramas se cercioran de que un ERD no contengan errores obvios. Para ayudarle a aplicar las reglas, ER Assistant proporciona una herramienta para revisar las reglas de los ERD terminados. Para ampliar sus bases sobre las notaciones de los ERD, este capítulo presentó las variaciones comunes que pudo haber encontrado, así como la notación de los diagramas de clases del Lenguaje de Modelado Unificado, una notación estándar del modelado orientado a objetos.

Este capítulo se enfocó en la notación de los ERD para proporcionar una sólida base para el estudio avanzado de la aplicación de la notación a los problemas de negocios. Para dominar el modelado de datos necesita comprender la notación ERD y obtener mucha práctica en la construcción de ERD. El capítulo 6 se enfoca en la práctica de la construcción de los ERD para los problemas de negocios. La aplicación de la notación incluye la representación consistente y completa de los requerimientos de los usuarios, generación de diseños alternativos y documentación de las decisiones del diseño. Además de estas habilidades, el capítulo 6 presenta las reglas para convertir un ERD en un diseño de tablas. Con el estudio cuidadoso del capítulo 5 y 6 obtendrá una base sólida para llevar a cabo el modelado de datos sobre las bases de datos de negocios.

## Revisión de conceptos

- Conceptos básicos: tipos de entidad, relaciones y atributos.
- Cardinalidades mínima y máxima para restringir la participación de relaciones.
- Existencia de la dependencia para las entidades que no pueden almacenarse sin el almacenamiento de entidades relacionadas.
- Dependencia identificable que involucra entidades débiles y relaciones identificables para respaldar los tipos de entidad que piden prestada al menos una parte de sus llaves primarias.
- Las relaciones M-N con atributos: los atributos están asociados con la combinación de tipos de entidad, y no sólo con uno de los tipos de entidad.
- Equivalencia entre una relación M-N y un tipo de entidad asociativo con relaciones 1-M identificables.
- Tipos de entidad asociativa M-way para representar las relaciones M-way entre más de dos tipos de entidades.
- Relaciones auto-referenciadas (unitarias) para representar asociaciones entre las entidades del mismo tipo de entidad.
- Diagramas de instancias para ayudar a distinguir entre las relaciones 1-M y M-N auto-referenciada.
- Jerarquías de generalización para mostrar las semejanzas entre los tipos de entidad.
- Representación de reglas de negocio en un ERD: identificación de entidades, conexiones entre entidades de negocios, número de entidades relacionadas, inclusión entre los conjuntos de entidades, valores razonables y colección de datos de integridad.
- Reglas de diagramas para prevenir errores obvios del modelado de datos.
- Fuente comunes de error de los diagramas: identificación de dependencias y de llaves foráneas redundantes.
- Respaldo para las reglas de diagramación con la herramienta ER Assistant.
- Variaciones en los ERD: símbolos y reglas de diagramación.
- Notación de diagramas de clases del lenguaje de modelado unificado como una alternativa al Modelo Entidad-Relación.

## Preguntas

1. ¿Qué es un tipo de entidad?
2. ¿Qué es un atributo?
3. ¿Qué es una relación?
4. ¿Cuál es la correspondencia del lenguaje natural para los tipos de entidad y las relaciones?
5. ¿Cuál es la diferencia entre un ERD y un diagrama de instancias?
6. ¿Qué símbolos son las contrapartes entre un ERD y un diagrama de instancias?
7. ¿Qué cardinalidades indican relaciones funcionales, opcionales y obligatorias?
8. ¿Cuándo es importante convertir una relación M-N en una relación 1-M?

9. ¿Cómo puede un diagrama de instancias ayudarle a determinar si una relación autorreferenciada es una relación 1-M o una relación M-N?
10. ¿Cuándo es que un ERD debe incluir entidades débiles?
11. ¿Cuál es la diferencia entre dependencia existencial y un tipo de entidad débil?
12. ¿Por qué es importante la clasificación en los negocios?
13. ¿Qué es la herencia dentro de las jerarquías de generalización?
14. ¿Cuál es el propósito de las restricciones de separación e integridad para las jerarquías de generalización?
15. ¿Qué símbolos se usan para la cardinalidad dentro de la notación de pata de cuervo?
16. ¿Cuáles son los dos componentes de la dependencia identifiable?
17. ¿Cómo se representan las relaciones M-way con la notación de pata de cuervo?
18. ¿Qué es un tipo de entidad asociativa?
19. ¿Cuál es la equivalencia entre una relación M-N y una relación 1-M?
20. ¿Qué significa decir que una parte de la llave primaria es prestada?
21. ¿Cuál es el propósito de las reglas de diagramación?
22. ¿Cuáles son las limitaciones de las reglas de diagramación?
23. ¿Qué reglas de consistencia violan con frecuencia los modeladores de datos novatos?
24. ¿Por qué los modeladores de datos novatos violan las reglas de la dependencia identifiable (reglas de consistencia de la 6 a la 8)?
25. ¿Por qué los modeladores de datos novatos violan la regla de consistencia 9 que trata sobre las llaves foráneas redundantes?
26. ¿Por qué una herramienta CASE debe respaldar las reglas de diagramación?
27. ¿De qué forma la herramienta ER Assistant respalda las reglas de consistencia 4 y 5?
28. ¿De qué forma la herramienta ER Assistant respalda todas las reglas excepto las reglas de consistencia 4 y 5?
29. ¿Por qué la herramienta ER Assistant no da solución a todos los errores de diagramación encontrados en un ERD?
30. ¿De qué forma la herramienta ER Assistant implementa la regla de consistencia 9, que trata sobre las llaves foráneas redundantes?
31. Liste algunas diferencias en los símbolos de la notación ERD que pudiese experimentar en su carrera.
32. Liste algunas diferencias en las reglas de diagramación de la notación ERD que pudiese experimentar en su carrera.
33. ¿Qué es el Lenguaje de Modelado Unificado (UML)?
34. ¿Cuáles son los elementos del modelado en un diagrama de clases de UML?
35. ¿Qué tipo de reglas de negocios se representan formalmente con la notación de pata de cuervo?
36. ¿Qué tipo de reglas de negocios están definidas mediante la documentación informal en ausencia de algún lenguaje de reglas para un ERD?

## Problemas

Los problemas se enfocan en el uso adecuado de la notación de la pata de cuervo y en la aplicación de las reglas de diagramación. Este enfoque es consistente con la pedagogía del capítulo. Los problemas más retadores del capítulo 6 se enfocan en los requerimientos de los usuarios, transformaciones de los diagramas, documentación del diseño y conversión de esquemas. Para mejorar su comprensión sobre el modelado de datos, debe resolver los problemas de los dos capítulos.

1. Dibuje un ERD que contenga los tipos de entidad *Order* y *Customer* conectados por una relación 1-M de *Customer* a *Order*. Elija un nombre apropiado para la relación utilizando su sentido común sobre las interacciones entre los clientes y las órdenes. Defina las cardinalidades mínimas para que una orden sea opcional para un cliente y un cliente sea obligatorio para una orden. Para el tipo de entidad *Customer*, agregue los atributos *CustNo* (llave primaria), *CustFirstName*, *CustLastName*, *CustStreet*, *CustCity*, *CustState*, *CustZip* y *CustBal* (Saldo). Para el tipo de entidad *Order*, agregue los atributos *OrdNo* (llave primaria), *OrdDate*, *OrdName*, *OrdStreet*, *OrdCity*, *OrdState* y *OrdZip*. Si está utilizando ER Assistant u otra herramienta de dibujo que soporte la especificación de tipos de datos, elija los tipos de datos apropiados para los atributos con base en su sentido común.

2. Amplíe el ERD del problema 1 con el tipo de entidad *Employee* y una relación 1-M de *Employee* a *Order*. Elija un nombre apropiado para la relación usando su sentido común en la interacción de empleados y órdenes. Defina las cardinalidades mínimas para que un empleado sea opcional en una orden y una orden sea opcional para un empleado. Para el tipo de entidad *Employee*, agregue los atributos *EmpNo* (llave primaria), *EmpFirstName*, *EmpLastName*, *EmpPhone*, *EmpEmail*, *EmpCommRate* (tasa de comisión) y *EmpDeptName*. Si está utilizando ER Assistant u otra herramienta que soporte la especificación de tipos de datos, escoja los tipos de datos apropiados para los atributos con base en su sentido común.
3. Amplíe el ERD del problema 2 con una relación autorreferenciada 1-M que incluya el tipo de entidad *Employee*. Seleccione un nombre apropiado para la relación utilizando sus conocimientos generales sobre las relaciones organizacionales entre los empleados. Defina las cardinalidades mínimas para que la relación sea opcional en ambas direcciones.
4. Amplíe el ERD del problema 3 con el tipo de entidad *Product* y una relación M-N entre *Product* y *Order*. Seleccione un nombre apropiado para la relación utilizando sus conocimientos generales sobre las conexiones entre los productos y las órdenes. Defina las cardinalidades mínimas para que una orden sea opcional en un producto, y para que un producto sea obligatorio en una orden. Para el tipo de entidad *Product*, agregue los atributos *ProdNo* (llave primaria), *ProdName*, *ProdQOH*, *ProdPrice* y *ProdNextShipDate*. Para la relación M-N, agregue un atributo para el monto de la orden. Si está utilizando ER Assistant o cualquier otra herramienta que soporte la especificación de tipos de datos, escoja los tipos de datos apropiados para los atributos con base en su sentido común.
5. Revise el ERD del problema 4 transformando la relación M-N en un tipo de entidad asociativa y dos relaciones 1-M identificables.



6. Revise los ERDs de los problemas 4 y 5 y busque violaciones a las reglas de diagramación. Si siguió las indicaciones del problema, sus diagramas no deben tener errores. Lleve a cabo la revisión sin utilizar la herramienta ER Assistant. Después, use la característica de revisión del diagrama de la herramienta ER Assistant.



7. Utilizando el ERD corregido del problema 6, agregue violaciones sobre las reglas de consistencia 6 a 9. Use la característica de revisión del diagrama de la herramienta ER Assistant para identificar errores.
8. Diseñe un ERD para el tipo de entidad *Task* y una relación M-N autorreferenciada. Para el tipo de entidad *Task*, agregue los atributos *TaskNo* (llave primaria), *TaskDesc*, *TaskEstDuration*, *TaskStatus*, *TaskStartTime* y *TaskEndTime*. Seleccione un nombre apropiado para la relación utilizando sus conocimientos generales sobre la precedencia de conexiones entre tareas. Defina las cardinalidades mínimas para que la relación sea opcional en ambas direcciones.
9. Revise el ERD del problema 8 transformando la relación M-N en un tipo de relación 1-M asociativa y dos identificables.
10. Defina una jerarquía de generalización que contenga los tipos de entidad *Student*, *UndStudent* y *GradStudent*. El tipo de entidad *Student* es el supertipo, y *UndStudent* y *GradStudent* los subtipos. El tipo de entidad *Student* tiene los atributos *StdNo* (llave primaria), *StdName*, *StdGender*, *StdDOB* (fecha de nacimiento), *StdEmail* y *StdAdmitDate*. El tipo de entidad *UndStudent* tiene los atributos *UndMajor*, *UndMinor* y *UndClass*. El tipo de entidad *GradStudent* tiene los atributos *GradAdvisor*, *GradThesisTitle* y *GradAsstStatus* (estatus del asistente). La jerarquía de generalización debe estar completa y separada.
11. Defina una jerarquía de generalización que contenga los tipos de entidad *Employee*, *Faculty* y *Administrator*. El tipo de entidad *Employee* es el supertipo y *Faculty* y *Administrator* los subtipos. El tipo de entidad *Employee* tiene los atributos *EmpNo* (llave primaria), *EmpName*, *EmpGender*, *EmpDOB* (fecha de nacimiento), *EmpPhone*, *EmpEmail* y *EmpHireDate*. El tipo de entidad *Faculty* tiene los atributos *FacRank*, *FacPayPeriods* y *FacTenure*. El tipo de entidad *Administrator* tiene los atributos *AdmTitle*, *AdmContractLength* y *AdmAppointmentDate*. La jerarquía de generalización debe estar completa y superposicionada.
12. Combine las jerarquías de generalización de los problemas 10 y 11. La raíz de la jerarquía de generalización es el tipo de entidad *UnivPerson*. La llave primaria de *UnivPerson* es *UnivSSN*. Los otros atributos del tipo de entidad *UnivPerson* deben ser atributos que sean comunes a *Employee* y *Student*. Debe renombrar los atributos para que sean consistentes con el tipo de entidad *UnivPerson*. La jerarquía de generalización debe estar completa y separada.
13. Dibuje un ERD que contenga los tipos de entidad *Patient*, *Physician* y *Visit*, conectadas por relaciones 1-M de *Patient* a *Visit* y de *Physician* a *Visit*. Elija los nombres apropiados para las relaciones. Defina las cardinalidades mínimas para que los pacientes y los médicos sean obligatorios para una visita, pero

que las visitas sean opcionales para pacientes y médicos. Para el tipo de entidad *Patient*, agregue los atributos *PatNo* (llave primaria), *PatFirstName*, *PatLastName*, *PatStreet*, *PatCity*, *PatState*, *PatZip* y *PatHealthPlan*. Para el tipo de entidad *Physician*, agregue los atributos *PhyNo* (llave primaria), *PhyFirstName*, *PhyLastName*, *PhySpeciality*, *PhyPhone*, *PhyEmail*, *PhyHospital* y *PhyCertification*. Para el tipo de entidad *Visit*, agregue los atributos *VisitNo* (llave primaria), *VisitDate*, *VisitPayMethod* (efectivo, cheque o tarjeta de crédito) y *VisitCharge*. Si está usando la herramienta ER Assistant o cualquier otra herramienta de dibujo que soporte la especificación de tipos de datos, elija los tipos de datos apropiados basándose en sus conocimientos generales.

14. Amplíe el ERD del problema 13 con los tipos de entidad *Nurse*, *Item* y *VisitDetail* conectadas por relaciones 1-M de *Visit* a *VisitDetail*, de *Nurse* a *VisitDetail* y de *Item* a *VisitDetail*. *VisitDetail* es una entidad débil con una relación 1-M de *Visit* a *VisitDetail* con relación identificable. Elija los nombres apropiados para las relaciones. Defina las cardinalidades mínimas para que la enfermera sea opcional para un detalle de visita, un elemento sea obligatorio para un detalle de visita y los detalles de vistas sean opcionales para las enfermeras y elementos. Para el tipo de entidad *Item*, agregue los atributos *ItemNo* (llave primaria), *ItemDesc*, *ItemPrice* y *ItemType*. Para el tipo de entidad *Nurse*, agregue los atributos *NurseNo* (llave primaria), *NurseFirstName*, *NurseLastName*, *NurseTitle*, *NursePhone*, *NurseSpeciality* y *NursePayGrade*. Para el tipo de entidad *VisitDetail* agregue los atributos *DetailNo* (parte de la llave primaria) y *DetailCharge*. Si está usando la herramienta ER Assistant o cualquier otra herramienta de dibujo que soporte la especificación de tipos de datos, elija los tipos de datos apropiados basándose en sus conocimientos generales.
15. Perfeccione el ERD del problema 14 con una jerarquía de generalización que incluya *Provider*, *Physician* y *Nurse*. La raíz de la jerarquía de generalización es el tipo de entidad *Provider*. La llave primaria de *Provider* es *ProvNo* que reemplaza a los atributos *PhyNo* y *NurseNo*. Los otros atributos del tipo de entidad *Provider* deben ser atributos comunes a *Nurse* y *Physician*. Debe renombrar los atributos para que sean consistentes con el tipo de entidad *Provider*. La jerarquía de generalización debe estar completa y separada.



16. Revise el ERD del problema 15 para encontrar violaciones a las reglas de diagramación. Si siguió las indicaciones del problema, su diagrama no debe tener errores. Aplique las reglas de consistencia e integridad para cerciorarse de que su diagrama no tenga errores. Si está usando la herramienta ER Assistant, puede usar la característica de revisión de diagramas para revisar las reglas usted mismo.



17. Utilizando el ERD corregido del problema 16, agregue violaciones a las reglas de consistencia 3 y 6 a 9. Si está usando las herramienta ER Assistant, puede usar la característica de revisión de diagramas después de haber revisado las reglas usted mismo.



18. Para cada error de consistencia de la figura 5.P1, identifique la regla de consistencia violada y sugiera posibles soluciones al error. El ERD tiene nombres genéricos para que pueda concentrarse en encontrar los errores del diagrama en lugar de enfocarse en el significado del diagrama. Si está usando la herramienta ER Assistant, puede comparar sus solución con el resultado usando la característica de revisión de diagramas.



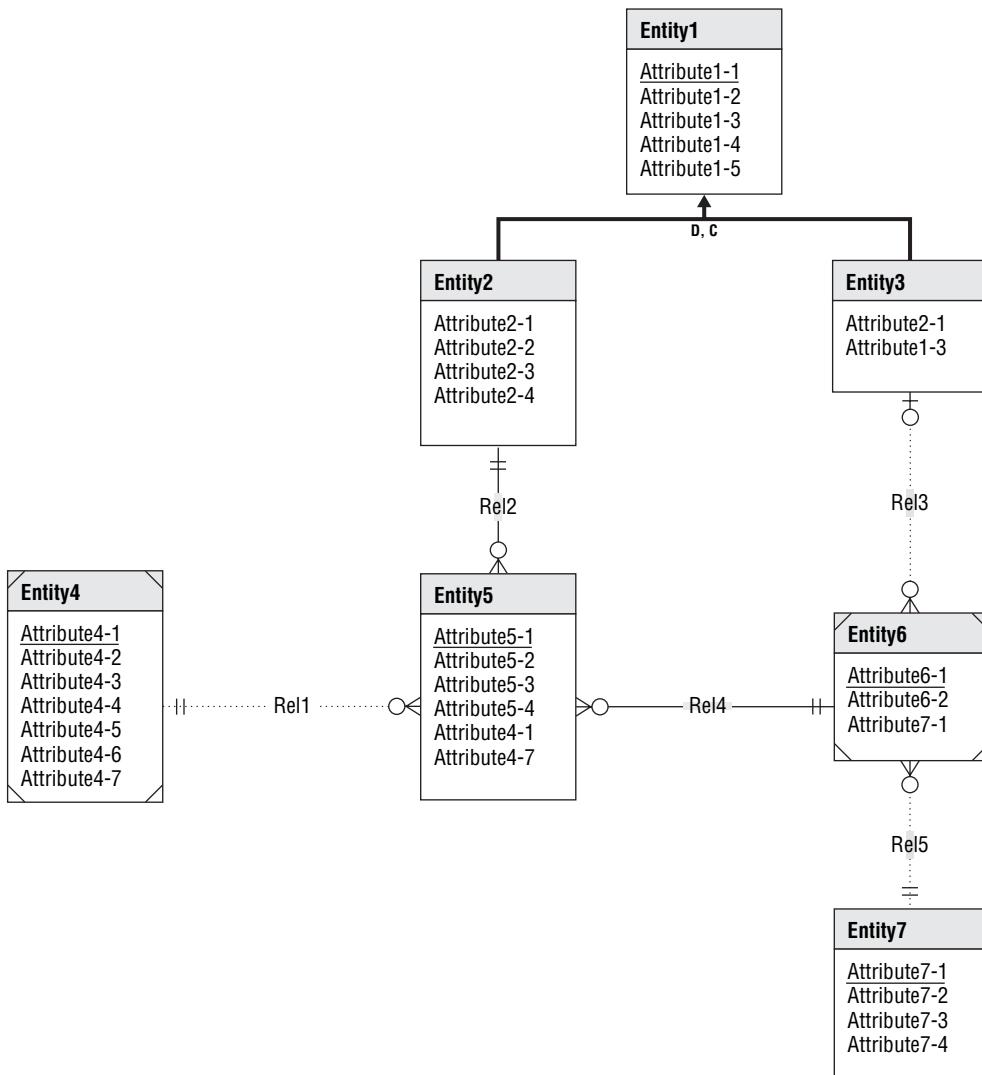
19. Para cada error de consistencia de la figura 5.P2, identifique la regla de consistencia violada y sugiera posibles soluciones al error. El ERD tiene nombres genéricos para que pueda concentrarse en encontrar los errores del diagrama en lugar de enfocarse en el significado del diagrama. Si está usando la herramienta ER Assistant, puede comparar sus solución con el resultado usando la característica de revisión de diagramas.



20. Para cada error de consistencia de la figura 5.P3, identifique la regla de consistencia violada y sugiera posibles soluciones al error. El ERD tiene nombres genéricos para que pueda concentrarse en encontrar los errores del diagrama en lugar de enfocarse en el significado del diagrama. Si está usando la herramienta ER Assistant, puede comparar sus solución con el resultado usando la característica de revisión de diagramas.

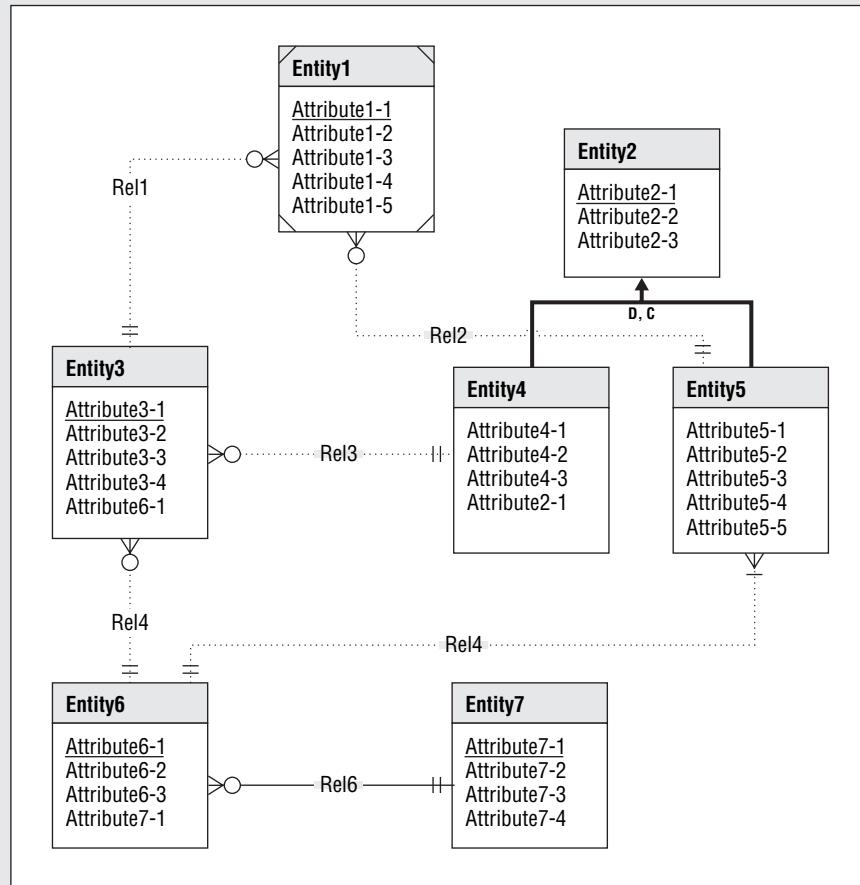
21. Dibuje un ERD que contenga los tipos de entidad *Employee* y *Appointment* conectados por una relación M-N. Elija un nombre apropiado para la relación usando sus conocimientos generales sobre las interacciones entre empleados y citas. Defina las cardinalidades mínimas para que una cita sea opcional para un empleado y un empleado sea obligatorio para una cita. Para el tipo de entidad *Employee*, agregue los atributos *EmpNo* (llave primaria), *EmpFirstName*, *EmpLastName*, *EmpPosition*, *EmpPhone* y *EmpEmail*. Para el tipo de entidad *Appointment*, agregue los atributos *AppNo* (llave primaria), *AppSubject*, *AppStartTime*, *AppEndTime* y *AppNotes*. Para la relación M-N, agregue un atributo *Attendance* que indique si el empleado fue a la cita. Si está utilizando la herramienta ER Assistant o cualquier otra herramienta de dibujo que soporte la especificación de tipos de datos, elija los tipos de datos apropiados basándose en sus conocimientos generales.

FIGURA 5.P1 ERD para el problema 18



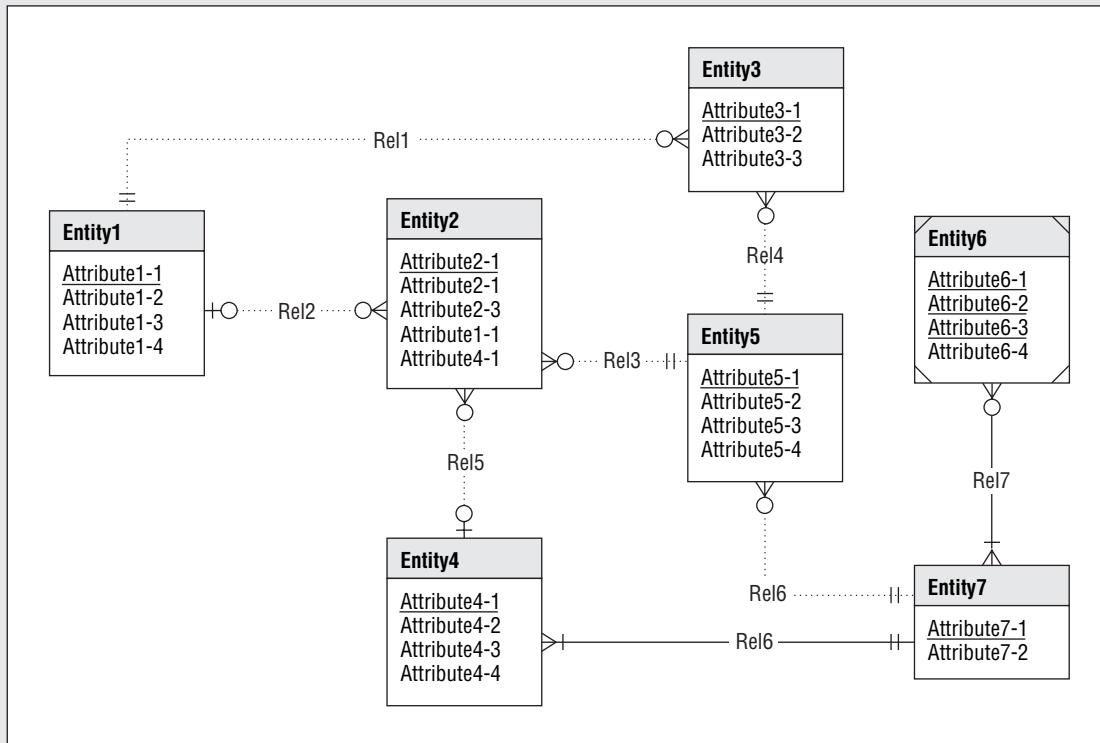
22. Amplíe el ERD del problema 21 con el tipo de entidad *Location* y una relación 1-M de *Location* a *Appointment*. Elija un nombre apropiado para la relación utilizando sus conocimientos generales sobre las interacciones entre ubicaciones y citas. Defina las cardinalidades mínimas para que una ubicación sea opcional para una cita y una cita sea opcional para una ubicación. Para el tipo de entidad *Location*, agregue los atributos *LocNo* (llave primaria), *LocBuilding*, *LocRoomNo*, y *LocCapacity*. Si está usando la herramienta ER Assistant o cualquier otra herramienta de dibujo que soporte la especificación de tipos de datos, elija los tipos de datos apropiados basándose en sus conocimientos generales.

**FIGURA 5.P2**  
ERD para el problema 19



23. Amplíe el ERD del problema 22 con el tipo de entidad *Calendar* y una relación M-N de *Appointment* a *Calendar*. Elija un nombre apropiado para la relación utilizando sus conocimientos generales sobre las interacciones entre citas y calendarios. Defina las cardinalidades mínimas para que una cita sea opcional para un calendario y un calendario sea obligatorio para una cita. Para el tipo de entidad *Calendar*, agregue los atributos *CalNo* (llave primaria), *CalDate* y *CalHour*. Si está usando la herramienta ER Assistant o cualquier otra herramienta de dibujo que soporte la especificación de tipos de datos, elija los tipos de datos apropiados basándose en sus conocimientos generales.
24. Revise el ERD del problema 23 y transforme la relación M-N entre *Employee* y *Appointment* en un tipo de entidad asociativa junto con dos relaciones 1-M identificables.

FIGURA 5.P3 ERD para el problema 20



## Referencias para ampliar su estudio

Cuatro libros especializados en el diseño de bases de datos son: Batini, Ceri y Navathe (1992); Nijssen y Haplin (1989); Teorey (1999), y Carlis y Maguire (2001). El sitio *DBAZine* ([www.dbazine.com](http://www.dbazine.com)) y el sitio de ayuda *DevX* ([www.devx.com](http://www.devx.com)) tienen muchos consejos prácticos sobre el desarrollo de bases de datos y modelado de datos. Si quiere saber más detalles sobre el UML, consulte el *Centro UML* ([umlcenter.visual-paradigm.com/index.html](http://umlcenter.visual-paradigm.com/index.html)) para revisar tutoriales y otras fuentes.

# Capítulo

# 6

---

# Desarrollo de modelo de datos para bases de datos de negocios

## Objetivos de aprendizaje

Este capítulo amplía su conocimiento sobre el modelado de datos desde la notación de los diagramas de entidad-relación (ERD) hasta el desarrollo de modelos de datos para bases de datos de negocios. Al finalizar este capítulo el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Desarrollar ERD que sean consistentes con los problemas narrativos.
- Utilizar transformaciones para generar ERD alternativos.
- Documentar decisiones de diseño implícitas en un ERD.
- Analizar un ERD para encontrar errores comunes del diseño.
- Convertir un ERD en un diseño de tablas mediante las reglas de conversión.

## Panorama general

---

El capítulo 5 explicó la notación de la pata de cuervo (*crow's foot*) para los diagramas de entidad-relación. Aprendió sobre los símbolos de los diagramas, patrones de relación, jerarquías de generalización y reglas de consistencia e integridad. La comprensión de la notación es un prerequisito para aplicarla a la representación de las bases de datos de negocios. Este capítulo explica el desarrollo de modelos de datos para bases de datos de negocios utilizando la notación de la pata de cuervo y las reglas para convertir un ERD en diseño de tablas.

Para convertirse en un buen modelador de datos, necesita comprender la notación de los diagramas de entidad-relación y adquirir mucha práctica en la construcción de diagramas. Este capítulo proporciona práctica en la aplicación de la notación. Aprenderá a analizar un problema narrativo, refinar el diseño por medio de transformaciones, documentar las decisiones de diseño más importantes y analizar un modelo de datos en búsqueda de errores comunes de diseño. Después de terminar un ERD, el diagrama debe convertirse en tablas relacionales para que se pueda implementar en un DBMS comercial. Este capítulo presenta las reglas para convertir un diagrama de entidad-relación en un diseño de tablas. Aprenderá las reglas básicas para transformar las partes comunes de un diagrama junto con reglas especializadas para las partes menos comunes.

Con estos antecedentes, usted estará listo para construir ERD para situaciones que se presentan en negocios de tamaño moderado. Adquirirá confianza en sus conocimientos de la notación de pata de cuervo, así como en la aplicación de la notación a problemas narrativos y en la conversión de diagramas en diseños de tablas.

## 6.1 Análisis de problemas de modelado de datos para negocios

---

Después de estudiar la notación de pata de cuervo, ahora está listo para aplicar su conocimiento. Esta sección presenta las guías para analizar las necesidades de información en los ambientes de negocios. Las guías incluyen el análisis de las descripciones de la narrativa del problema, así como los retos de los requerimientos para determinar la información, incluso en situaciones en las que los negocios no estén estructurados. Después de presentar las guías, se aplicarán al desarrollo de un ERD como ejemplo de un problema de modelado de datos de un negocio.

### 6.1.1 Guías para analizar las necesidades de información de los negocios

El modelado de datos abarca la obtención y el análisis de los requerimientos del negocio para generar un ERD que represente dichos requerimientos. Los requerimientos de los negocios muy pocas veces están bien estructurados. En su papel de analista comúnmente tendrá que enfrentar una situación de negocios sin definir, la cual usted tendrá que estructurar. Tendrá que interactuar con varios de los interesados, quienes algunas veces proporcionan indicaciones acerca de los requerimientos de la base de datos. Al recolectar los requerimientos, realizará entrevistas, revisará documentos y documentación de sistemas y examinará datos actuales. Para determinar el alcance de la base de datos, necesitará eliminar detalles irrelevantes y agregar detalles que hagan falta. Para proyectos grandes, necesitará trabajar sobre un subconjunto de requerimientos y después colaborar con un grupo de diseñadores para determinar el modelo de datos completo.

Estos retos hacen que el modelado de datos sea una actividad estimulante e intelectual. Un modelo de datos proporciona un elemento esencial para estandarizar el vocabulario de la organización, obligar a que se usen reglas de negocios y asegurar una adecuada calidad de datos. Muchos usuarios experimentarán los resultados de su esfuerzo mientras utilicen la base de datos en el trabajo diario. Debido a que los datos electrónicos han venido a ser un recurso corporativo vital, sus esfuerzos sobre el modelado de datos pueden crear una diferencia significativa en el futuro éxito de una organización.

Un libro de texto no puede proporcionar la experiencia en el diseño de bases de datos reales. Los problemas más difíciles del capítulo y los casos de estudio asociados en el sitio web del curso pueden proporcionar bosquejos de las dificultades que enfrentará en el diseño de bases de datos reales, pero no le proporcionarán la práctica obtenida con la experiencia actual. Para adquirir experiencia debe interactuar con organizaciones a través de proyectos de clases, internados y experiencia laboral. Por ende, este capítulo se enfoca en las metas más limitadas sobre análisis de problemas narrativos como un paso para desarrollar sus habilidades en el modelado de datos para situaciones reales de negocios. El análisis de problemas narrativos le ayudará a adquirir confianza al traducir un enunciado de problema en un ERD, e identificar las partes ambiguas e incompletas de los enunciados de un problema.

El objetivo principal al analizar los problemas narrativos es crear un ERD que sea consistente con la narrativa. El ERD no debe contradecir los elementos ERD implícitos en la narrativa del problema. Por ejemplo, si el enunciado del problema señala que los conceptos están relacionados mediante palabras que indican más de uno, el ERD debe tener la cardinalidad de muchos para que coincida con esa parte del problema. El resto de esta sección y de la sección 6.3.2 proporciona más detalles acerca de cómo conseguir que un ERD sea consistente.

Además del objetivo de consistencia, usted debe tener una tendencia hacia los diagramas más sencillos en lugar de los más complejos. Por ejemplo, un ERD con un solo tipo de entidad es menos complejo que un ERD con dos tipos de entidad y una relación. En general, cuando exista

#### **objetivo del análisis de problemas narrativos**

procurar un diseño simple que sea consistente con la narrativa. Prepararlo para darle seguimiento con requerimientos adicionales y la consideración de diseños alternativos.

una elección entre dos ERD, usted debe escoger el diseño más simple, especialmente en las etapas iniciales del proceso de diseño. Conforme progrese el proceso de diseño puede agregar detalles y refinamientos al diseño original. La sección 6.2 proporciona una lista de transformaciones que le pueden ayudar a considerar diseños alternativos.

### *Identificación de los tipos de entidad*

En una narrativa, debe buscar los sustantivos que involucren personas, cosas, lugares y eventos como potenciales tipos de entidad. Los sustantivos pueden aparecer como sujetos u objetos en los enunciados. Por ejemplo, el enunciado “Los estudiantes toman cursos en la universidad” indica que los estudiantes y los cursos pueden ser tipos de entidad. Debe buscar sustantivos que tengan enunciados adicionales que describan sus propiedades. Las propiedades generalmente señalan los atributos de un tipo de entidad. Por ejemplo, el enunciado “Los estudiantes eligen su carrera y especialidad en el primer año” indica que la carrera y la especialidad tal vez sean atributos del estudiante. El enunciado “Los cursos tienen un número de curso, semestre, año y salón listado en el catálogo” indica que el número de curso, semestre, año y salón tal vez sean atributos de curso.

El principio de simplicidad debe aplicarse durante la búsqueda de los tipos de entidad en el ERD inicial, en especial, las selecciones que involucren atributos y tipos de entidad. A menos que la descripción del problema contenga enunciados adicionales o detalles de un sustantivo, debe considerarlo inicialmente como un atributo. Por ejemplo, si los cursos tienen el nombre de un instructor listado en el catálogo, debe considerar el nombre del instructor como un atributo del tipo de entidad curso en lugar de un tipo de entidad, a menos que se proporcionen detalles adicionales sobre los instructores en el enunciado del problema. Si existe confusión entre considerar un concepto como un atributo o un tipo de entidad, debe continuar con los siguientes requerimientos.

### *Identificación de llaves primarias*

La identificación de llaves primarias es una parte importante de la identificación del tipo de entidad. Lo ideal es que las llaves primarias sean estables y tengan un propósito único. “Estable” significa que una llave primaria nunca debe cambiar después de haberla asignado a una entidad. “Propósito único” significa que un atributo de llave primaria no debe tener otro propósito distinto a la identificación de la entidad. Típicamente, los valores enteros generados de forma automática por un DBMS son buenas elecciones para llaves primarias. Por ejemplo, *Access* tiene un tipo de entidad *AutoNumber* para las llaves primarias y *Oracle* tiene el objeto *Sequence* para las llaves primarias.

Si los requerimientos indican la llave primaria para un tipo de entidad, debe asegurarse de que la llave primaria propuesta sea estable y con un único propósito. Si la llave primaria propuesta no cumple con ninguno de los criterios, probablemente deba rechazarla como llave primaria. Si la llave primaria propuesta sólo cumple uno de los criterios, debe explorar otros atributos para la llave primaria. En ocasiones, las prácticas de la industria o de la organización dictan la llave primaria incluso si ésta no es la ideal.

Además de las llaves primarias, también debe identificar otros atributos únicos (llaves candidatas). Por ejemplo, la dirección de correo electrónico de un empleado generalmente es única. La integridad de las llaves candidatas puede ser importante para búsquedas e integración con bases de datos externas. Dependiendo de las características de la herramienta de dibujo del ERD que utilice debe observar que un atributo es único, ya sea en la especificación del atributo o en la documentación. Las restricciones unitarias se pueden aplicar después de haber transformado un ERD en diseño de tablas.

### *Agregando relaciones*

Las relaciones generalmente aparecen como verbos que conectan a los sustantivos que previamente se identificaron como tipos de entidad. Por ejemplo, el enunciado “Los estudiantes se inscriben a cursos cada semestre” indica una relación entre los estudiantes y los cursos. Para la cardinalidad de la relación, debe observar el número (singular o plural) de sustantivos junto con otras palabras que indiquen la cardinalidad. Por ejemplo, el enunciado “Cada curso que se ofrece

es impartido por un instructor” indica que existe un instructor por cada curso que se ofrece. También debe buscar palabras como “colección” y “conjunto” que indican una cardinalidad máxima de más de uno. Por ejemplo, el enunciado “Una orden contiene una colección de elementos” indica que una orden está relacionada con muchos elementos. La cardinalidad mínima puede estar indicada por palabras como “opcional” y “requerido”. En la ausencia de indicaciones sobre la cardinalidad mínima, deben considerarse los valores por omisión. Deben realizarse los requerimientos adicionales de recopilación para confirmar los valores por omisión.

Debe estar alerta con las indicaciones de las relaciones en los enunciados de problemas, ya que pueden conducirlo a conexiones directas o indirectas en un ERD. Una conexión directa involucra una relación entre los tipos de entidad. Una conexión indirecta involucra una conexión a través de otros tipos de entidad y relaciones. Por ejemplo, el enunciado “Un orientador aconseja a los estudiantes acerca de la elección de una carrera” puede indicar relaciones directas o indirectas entre orientador, estudiante y carrera.

Para resolver las elecciones difíciles entre las conexiones directas e indirectas debe buscar los tipos de entidad que estén involucrados en relaciones múltiples. Estos tipos de entidad pueden reducir el número de relaciones en un ERD al colocarlos en el centro como un concentrador conectado directamente con otros tipos de entidad, como los rayos de una rueda. Los tipos de entidad derivados de documentos importantes (órdenes, registros, órdenes de compra, etc.) generalmente se concentran en un ERD. Por ejemplo, un tipo de entidad orden se puede relacionar de forma directa con un cliente, empleado y producto eliminando la necesidad de conexiones directas entre todos los tipos de entidad. Estas opciones se remarcarán en el análisis de requerimientos de información de la compañía de servicios de agua de la siguiente sección.

### *Resumen de las guías de análisis*

Cuando se analiza el enunciado de un problema narrativo, debe desarrollar un ERD que de forma consistente represente la narrativa completa. Dada una opción entre los ERD consistentes, usted debe estar en favor de los diseños más simples en lugar de los más complejos. También

**TABLA 6.1 Resumen de las guías de análisis para problemas narrativos**

Elemento del diagrama	Guías de análisis	Efecto ERD
Identificación del tipo de entidad	Busque los sustantivos utilizados como sujetos u objetos solos, con detalles adicionales en otros enunciados.	Agregue los tipos de entidad al ERD. Si el sustantivo no agrega detalles adicionales, considérelo como un atributo.
Determinación de la llave primaria	Busque los atributos estables y de propósito único para las llaves primarias. La narrativa debe indicar que son únicos.	Especifique las llaves primarias y candidatas.
Detección de la relación (directa o indirecta)	Busque los verbos que conecten a los sustantivos identificados como tipos de entidad.	Agregue la relación directa entre los tipos de entidad o identifique la existencia de una conexión entre los tipos de entidad.
Determinación de la cardinalidad (máxima)	Busque la designación singular o plural de los sustantivos en los enunciados que señalan relaciones.	Especifique cardinalidades de 1 y M (muchos).
Determinación de la cardinalidad (mínima)	Busque los términos opcionales o requeridos en los enunciados. Identifique requerido como el valor por omisión si el enunciado del problema no indica la cardinalidad mínima.	Especifique cardinalidades de 0 (opcional) y 1 (obligatorio).
Simplificación de la relación	Busque los tipos de entidad concentradora como sustantivos utilizados en varias oraciones vinculadas a otros sustantivos identificados como tipos de entidad.	Un tipo de entidad concentradora tiene relaciones directas con otros tipos de entidad. Elimine otras relaciones si existe una conexión directa a través del tipo de entidad concentradora.

debe observar las ambigüedades y elementos que no estén completos en el enunciado del problema. Las guías descritas en esta sección pueden ayudarle en su análisis inicial sobre problemas de modelado de datos. Las secciones 6.2 y 6.3 presentan métodos adicionales de análisis para revisar y terminar los ERD. La tabla 6.1 presenta un resumen con el fin de ayudarle a recordar las guías descritas en esta sección.

### 6.1.2 Análisis de los requerimientos de información para la base de datos de la compañía de servicios de agua

Esta sección presenta los requerimientos para una base de datos de clientes de una compañía municipal de servicios de agua. Usted puede asumir que esta descripción es el resultado de una investigación inicial con el personal apropiado de la compañía de agua. Después de la descripción, las guías presentadas en la sección 6.1.1 se utilizan para analizar la descripción de la narrativa y desarrollar un ERD.

#### *Requerimientos de información*

La base de datos de la compañía de servicios de agua debe respaldar el registro y la facturación del uso de agua. Para apoyar estas funciones, la base de datos debe tener datos de clientes, tarifas uso del agua y facturas. En esta descripción se omiten otras funciones como el procesamiento de los pagos y el servicio a clientes. La siguiente lista describe los requerimientos de datos con más detalle.

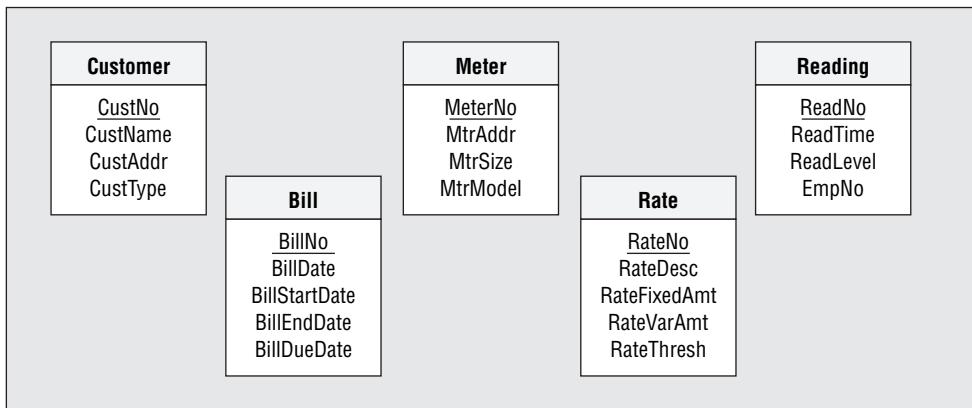
- Los datos de los clientes incluyen un número de cliente único, nombre, dirección de facturación, tipo (comercial o residencial), tarifa y conjunto de medidores (1 o más).
- Los datos de los medidores incluyen número único de medidor, dirección, tamaño y modelo. El número del medidor se graba en el mismo antes de ponerlo en servicio. Un medidor está asociado con un cliente a la vez.
- Un empleado periódicamente lee cada medidor en una fecha determinada. Cuando se lee un medidor, se crea un documento que contiene un número único de lectura del medidor, un número de empleado, un número de medidor, un tiempo (incluye fecha y hora) y un nivel de consumo. Cuando un medidor se pone en servicio por primera vez, no hay ninguna lectura asociada a él.
- Una tarifa incluye un número único de tarifa, una descripción, un monto fijo en dólares, un límite de consumo y un monto variable (dólares por metro cúbico). El consumo por arriba del límite se factura como monto variable. A los clientes se les asignan tarifas utilizando varios factores, como tipo de cliente, dirección y factores de ajuste. A muchos clientes se les puede asignar la misma tarifa. Las tarifas típicamente se proponen meses antes de ser aprobadas y se asocian con los clientes.
- Las facturas por uso de agua están basadas en las más recientes lecturas de los medidores de los clientes y en las tarifas que les aplica. Una factura está formada por un encabezado y una lista de líneas de detalle. El encabezado contiene un folio de factura, un número de cliente, una fecha de facturación, una fecha límite de pago y un rango de fechas del periodo de consumo. Cada línea de detalle contiene número de medidor, nivel de consumo de agua y monto. El nivel de consumo de agua se calcula restando los niveles de consumo en las dos lecturas más recientes del medidor. El monto se calcula al multiplicar el nivel de consumo por la tarifa del cliente.

#### *Identificación de los tipos de entidad y llaves primarias*

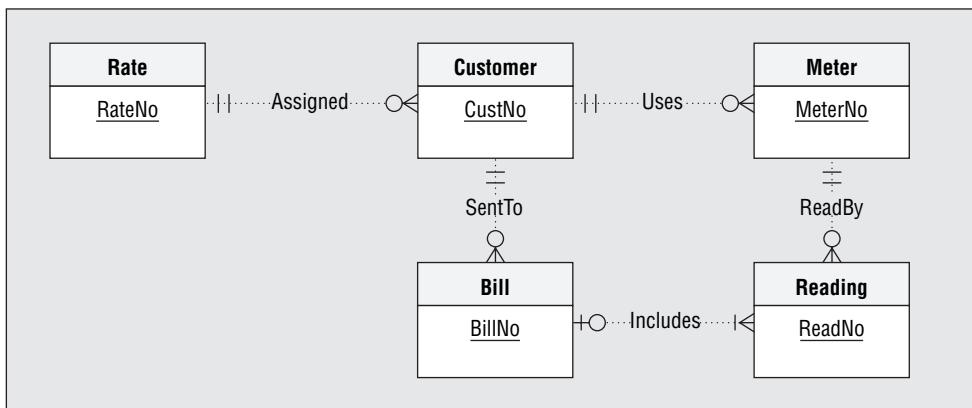
Los sustantivos importantes de la narrativa son cliente, medidor, factura, lectura y tarifa. Para cada uno de estos sustantivos la narrativa describe atributos asociados. La figura 6.1 muestra un ERD preliminar con los tipos de entidad para los sustantivos y atributos asociados. Observe que los conjuntos de cosas no son atributos. Por ejemplo, el hecho de que un cliente tenga un conjunto de medidores se mostrará como una relación, en lugar de un atributo para el tipo de entidad *Customer*. Además, las referencias entre estos tipos de entidad se mostrarán como relaciones en lugar de atributos. Por ejemplo, el hecho de que una lectura contenga un número de medidor se registrará como una relación.

**FIGURA 6.1**

Tipos de entidad preliminar y atributos de la base de datos de la compañía de servicios de agua

**FIGURA 6.2**

Tipos de entidad conectadas mediante relaciones



La narrativa menciona de forma específica la existencia única de los números de cliente, de medidor, de lectura, de factura y de tarifa. El número de factura, número de lectura y número de medidor parecen ser estables y de propósito único, ya que están impresos en objetos físicos.

Se deben realizar investigaciones adicionales para determinar si el número de un cliente y el número de la tarifa son estables y de propósito único. Debido a que la narrativa no describe los usos adicionales de estos atributos, el supuesto inicial dentro del ERD es que estos atributos pueden funcionar como llaves primarias.

#### *Adición de relaciones*

Después de identificar los tipos de entidad y atributos, continuemos conectando los tipos de entidad con las relaciones, tal como se muestra en la figura 6.2. Para reducir el tamaño del ERD, en la figura 6.2 sólo se muestran las llaves primarias. Un buen lugar para comenzar es con las partes de la narrativa que señalan las relaciones entre los tipos de entidad. La siguiente lista explica la deducción de las relaciones a partir de la narrativa.

- Para la relación *Assigned*, la narrativa establece que un cliente tiene una tarifa y a muchos clientes se les puede asignar la misma tarifa. Estos dos enunciados indican una relación 1-M de *Rate* a *Customer*. Para las cardinalidades mínimas, la narrativa señala que se requiere una tarifa para un cliente, y que las tarifas se proponen antes de asociarlas con los clientes.
- Para la relación *Uses*, la narrativa señala que un cliente incluye una colección de medidores y que un medidor está asociado con un sólo cliente a la vez. Estos dos enunciados señalan una relación 1-M de *Customer* a *Meter*. Para las cardinalidades mínimas, la narrativa señala que un cliente debe tener al menos un medidor. La narrativa no indica la cardinalidad mínima

para un medidor, así que se puede escoger 0 o 1. La documentación debe identificar en las especificaciones estos elementos incompletos.

- Para la relación *ReadBy*, la narrativa establece que la lectura de un medidor contiene un nombre de medidor y que los medidores se leen periódicamente. Estos dos enunciados indican una relación 1-M de *Meter* a *Reading*. Para las cardinalidades mínimas, la narrativa indica que se requiere un medidor para una lectura, y que un medidor nuevo no tiene ninguna lectura asociada.
- Para la relación *SentTo*, la narrativa indica que el encabezado de una factura contiene un número de cliente y que las facturas se envían periódicamente a los clientes. Estos dos enunciados señalan una relación 1-M de *Customer* a *Bill*. Para las cardinalidades mínimas, la narrativa indica que se requiere un cliente para una factura, y que un cliente no tiene asociada ninguna factura hasta que se leen los medidores del cliente.

La relación *Includes* entre los tipos de entidad *Bill* y *Reading* es sutil. La relación *Includes* es 1-M, ya que una factura puede incluir un conjunto de lecturas (una en cada línea de detalle), y una lectura se relaciona con una factura. El nivel de consumo y el monto en una línea de detalle son valores calculados. La relación *Includes* conecta una factura con las lecturas más recientes del medidor, por lo tanto, respalda los cálculos del consumo y el monto. Estos valores se pueden almacenar porque cuando se les requiera resulta más eficiente almacenarlos que calcularlos. Si los valores se almacenan, se pueden añadir los atributos a la relación *Includes* o al tipo de entidad *Reading*.

## 6.2 Refinamientos a un ERD

---

El modelado de datos es generalmente un proceso iterativo o repetitivo. Usted construye un modelo de datos preliminar y después lo refina varias veces. Al refiniar un modelo de datos debe generar alternativas factibles y evaluarlas de acuerdo con los requerimientos de los usuarios. Especialmente necesita obtener información adicional de los usuarios para evaluar las alternativas. Este proceso de refinamiento y evaluación puede continuar muchas veces para grandes bases de datos. Para ilustrar la naturaleza iterativa del modelado de datos, esta sección describe algunos refinamientos que se puedan hacer al diseño inicial del ERD de la figura 6.2.

### 6.2.1 Transformación de los atributos en tipos de entidad

Un refinamiento común es transformar un atributo en un tipo de entidad. Esta transformación es útil cuando la base de datos debe contener más que simplemente el identificador de una unidad. Esta transformación incluye la adición de un tipo de entidad y de una relación 1-M. En el ERD de la compañía de agua, el tipo de entidad *Reading* contiene el atributo *EmpNo*. Si se necesitan otros datos de un empleado, se puede ampliar *EmpNo* en un tipo de entidad y en una relación 1-M, como se muestra en la figura 6.3.

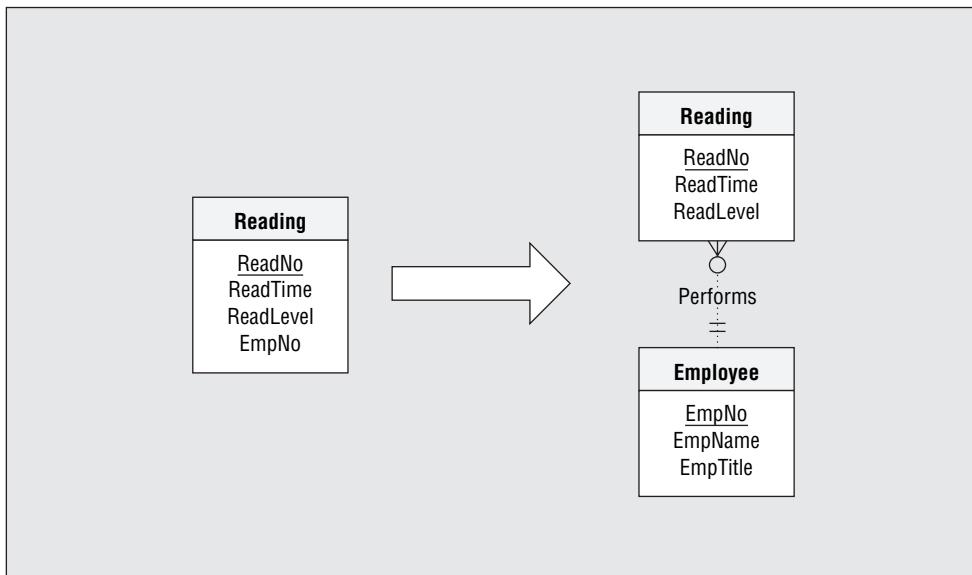
### 6.2.2 División de atributos compuestos

Otro refinamiento común es dividir los atributos compuestos en atributos más pequeños. Un atributo compuesto contiene muchos tipos de datos. Por ejemplo, el tipo de entidad *Customer* tiene un atributo dirección que contiene datos acerca de la calle del cliente, ciudad, estado y código postal. La división de los atributos compuestos puede facilitar la búsqueda de los datos inmersos. La división del atributo dirección como se muestra en la figura 6.4 soporta búsquedas por calle, ciudad, estado y código postal.

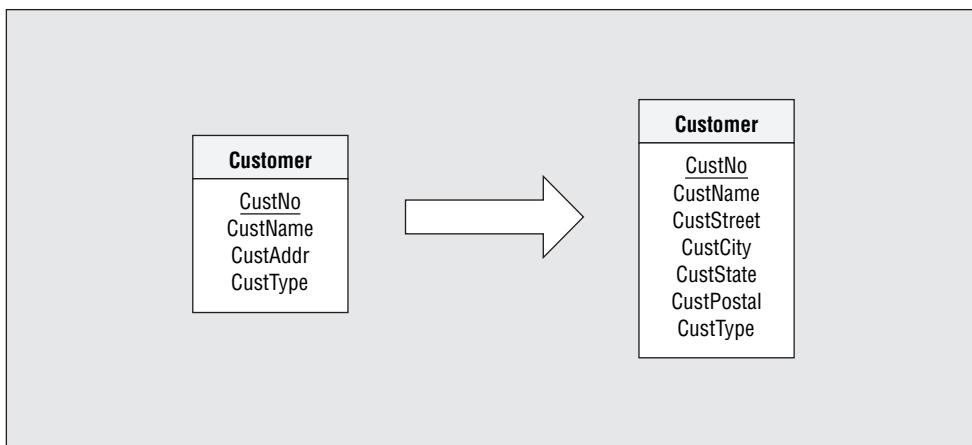
### 6.2.3 Expansión de los tipos de entidad

Una tercera transformación es dividir un tipo de entidad en dos tipos de entidad y en una relación. Esta transformación puede ser útil para registrar un nivel más fino de detalle sobre una entidad. Por ejemplo, las tarifas en la base de datos de la compañía de agua se aplican a todos los niveles de consumo más allá de un nivel fijo. Puede ser útil tener una estructura de tarifas más

**FIGURA 6.3**  
Transformación de  
un atributo en un tipo  
de entidad



**FIGURA 6.4**  
División del atributo  
*CustAddr* en sus com-  
ponentes

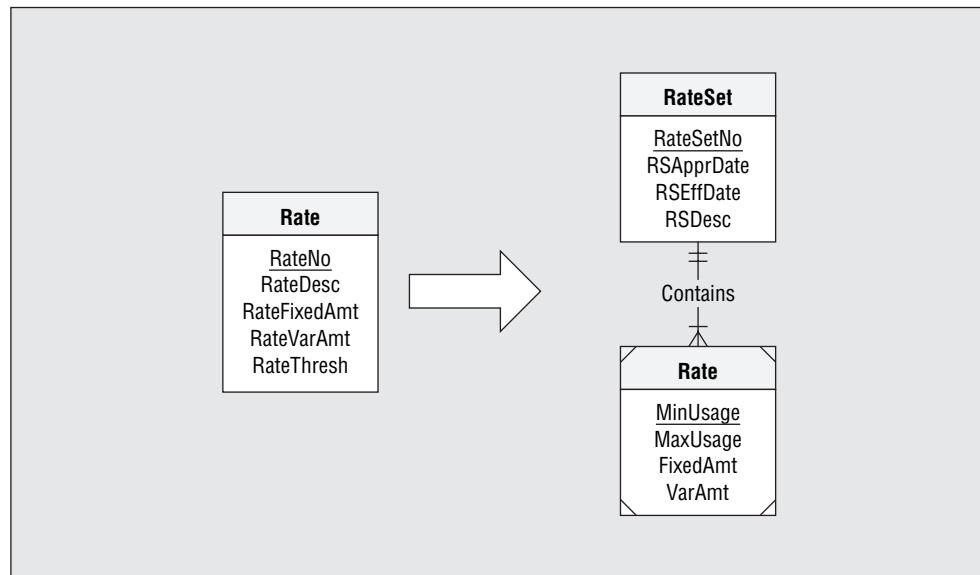


complejas, en la cual el monto variable dependa del nivel de consumo. La figura 6.5 muestra una transformación hacia el tipo de entidad *Rate* para respaldar una estructura de tarifas más compleja. El tipo de entidad *RateSet* representa un conjunto de tarifas aprobado por la comisión de gobierno de la compañía. La llave primaria del tipo de entidad *Rate* se genera a partir del tipo de entidad *RateSet*. La identificación de dependencias no se requiere cuando se transforma un tipo de entidad en dos tipos de entidad y en una relación. En esta situación, la identificación de dependencias es útil, pero en otras situaciones puede que no sea apropiada.

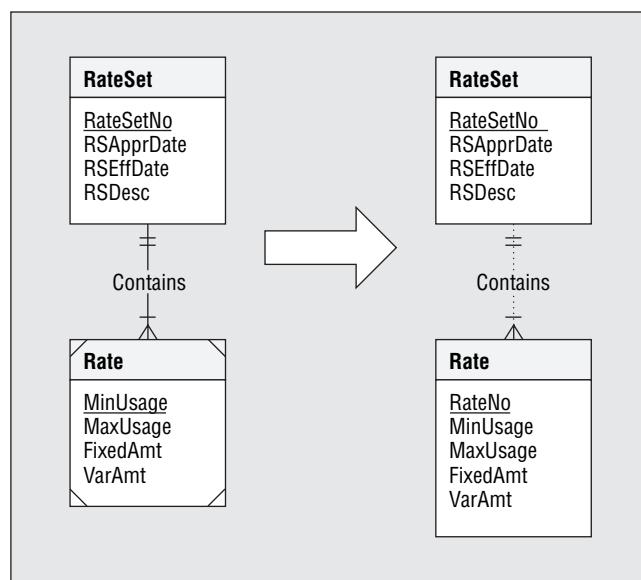
#### 6.2.4 Transformación de una entidad débil en una entidad fuerte

Una cuarta transformación es hacer de una entidad débil una entidad fuerte y modificar las relaciones asociadas identificables en relaciones no identificables. Esta transformación puede hacer que sea más fácil referenciar a un tipo de entidad después de hacer la conversión al diseño de tablas. Después de la conversión, una referencia a una entidad débil incluirá una llave foránea combinada con más de una columna. Esta transformación es muy útil para los tipos de entidad asociativos, especialmente para los tipos de entidad asociativos que representan relaciones M-way.

**FIGURA 6.5**  
Transformación de un tipo de entidad en dos tipos de entidad y una relación



**FIGURA 6.6**  
Transformación de una entidad débil en una entidad fuerte

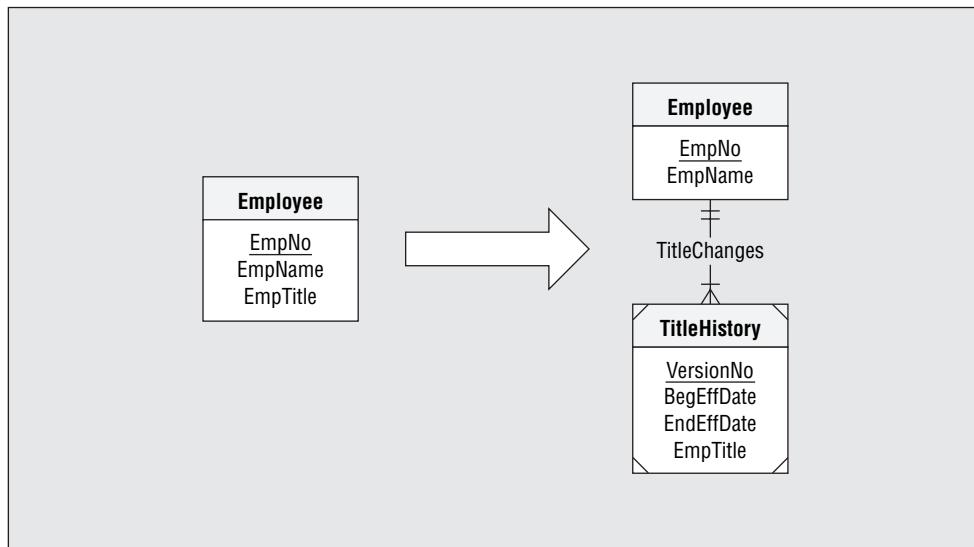


La figura 6.6 ilustra la transformación de la entidad débil *Rate* en una entidad fuerte. La transformación incluye la modificación de una entidad débil en una entidad fuerte y la modificación de cada relación identificable en una relación no identificable. Además, puede ser necesario agregar un nuevo atributo que funcione como la llave primaria. En la figura 6.6 el nuevo atributo *RateNo* es la llave primaria, ya que *MinUsage* no identifica las tarifas de forma única. El diseñador debe observar que la combinación de *RateSetNo* y *MinUsage* es única en la documentación del diseño, así que una restricción de una llave candidata se puede especificar después de la conversión a un diseño de tablas.

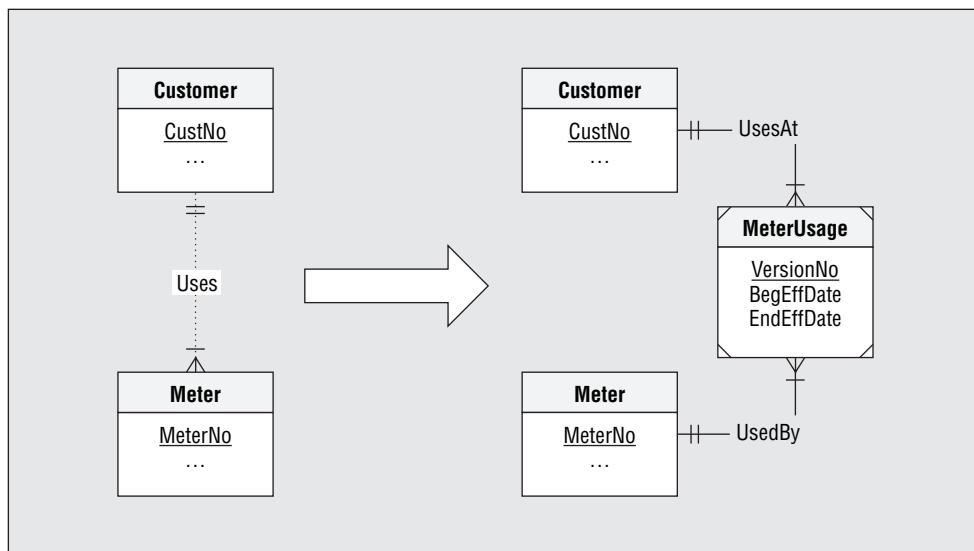
### 6.2.5 Agregando historia

Una quinta transformación es agregar detalles históricos a un modelo de datos. Los detalles históricos pueden ser necesarios para requerimientos legales, así como para requerimientos de reportes estratégicos. Esta transformación se puede aplicar a los atributos y a las relaciones.

**FIGURA 6.7**  
Agregando historia al atributo *EmpTitle*



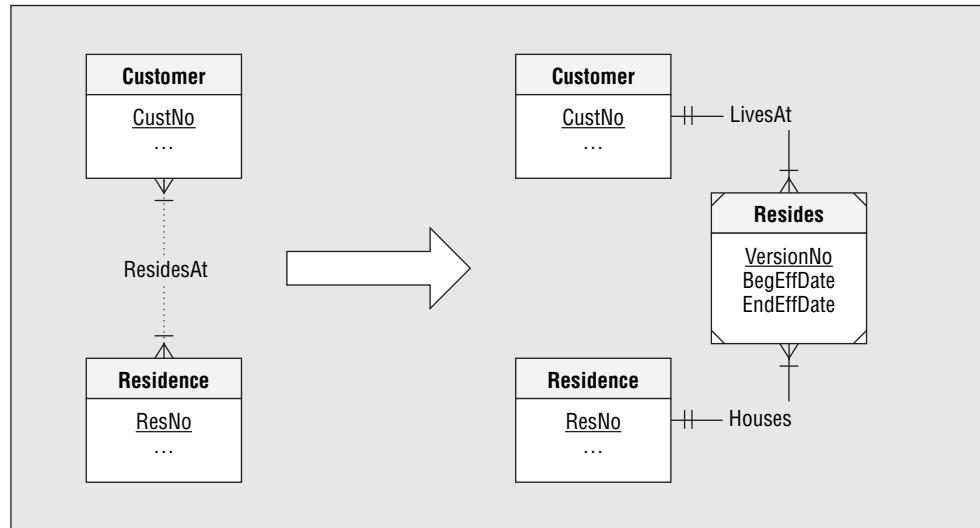
**FIGURA 6.8**  
Añadiendo historia a una relación 1-M



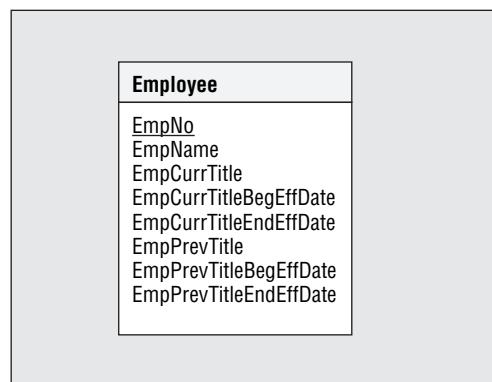
Cuando se aplica a los atributos, la transformación es similar al atributo que transforma el tipo de entidad. Por ejemplo, para conservar una historia de los puestos de los empleados, se reemplaza el atributo *EmpTitle* por un tipo de entidad y una relación 1-M. El nuevo tipo de entidad típicamente contiene un número de versión como parte de su llave primaria y el resto de su llave primaria lo obtiene del tipo de entidad original, tal como se muestra en la figura 6.7. Las fechas iniciales y finales indican las fechas efectivas de la modificación.

Cuando se aplica a una relación, esta transformación típicamente incluye la modificación de una relación 1-M en un tipo de entidad asociativo y en un par de relaciones identificables 1-M. La figura 6.8 ilustra la transformación de la relación 1-M *Uses* en un tipo de entidad asociativo con atributos para el número de versión y para las fechas efectivas. El tipo de entidad asociativo es necesario porque la combinación de cliente y medidor quizás no sea única sin el número de versión. Cuando se aplica a una relación M-N, esta transformación involucra un resultado similar. La figura 6.9 ilustra la transformación de la relación M-N *ResidesAt* en un tipo de entidad asociativa con un número de versión y cambio efectivo del atributo de fecha.

**FIGURA 6.9**  
Añadiendo historia a una relación M-N



**FIGURA 6.10**  
Añadiendo historia limitada al tipo de entidad *Employee*



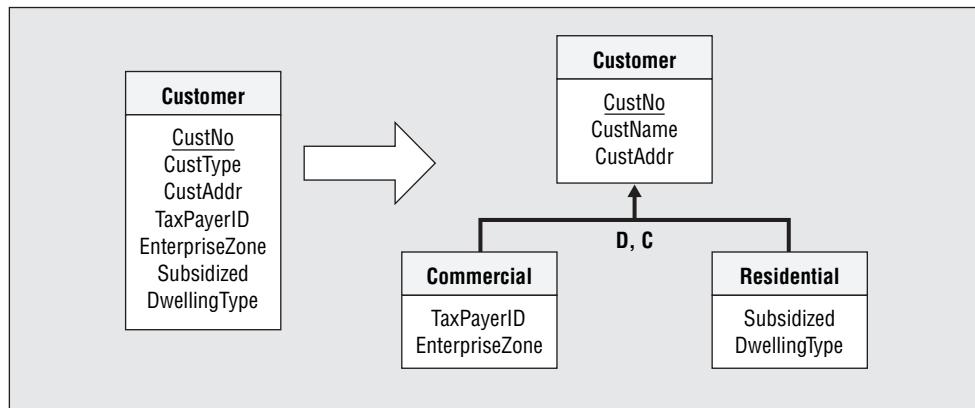
Las transformaciones de las figuras 6.7 a 6.9 soportan una historia ilimitada. Para una historia limitada se puede agregar un número fijo de atributos al mismo tipo de entidad. Por ejemplo, para conservar la historia de los puestos del empleado actual y más reciente, es posible usar dos atributos (*EmpCurrTitle* y *EmpPrevTitle*) como se ilustra en la figura 6.10. Para registrar las fechas de modificación de los puestos de los empleados, se pueden agregar dos atributos de fecha efectivos para cada atributo de puesto.

### 6.2.6 Añadiendo jerarquías de generalización

Una sexta transformación es hacer de un tipo de entidad una jerarquía de generalización. Esta transformación se debe hacer pocas veces, ya que la jerarquía de generalización es una herramienta especializada de modelado. La jerarquía de generalización puede ser útil si existen varios atributos que no se utilicen en todas la entidades y además existe una clasificación aceptada de las entidades. Por ejemplo, los clientes de la compañía de agua se pueden clasificar en comerciales o residenciales. Los atributos específicos de los clientes comerciales (*TaxPayerID* y *EnterpriseZone*) no aplican para los clientes residenciales y viceversa. En la figura 6.11 los atributos específicos de los clientes comerciales y residenciales se han movido hacia los subtipos. Un beneficio adicional de esta transformación es que se evitan los valores nulos. Por ejemplo, las entidades de los tipos de entidad *Commercial* y *Residential* no tendrán valores nulos. En el tipo de entidad original *Customer*, los clientes residenciales no tendrían valores nulos para

**FIGURA 6.11**

Transformación de la jerarquía de generalización para los clientes de la compañía de agua

**TABLA 6.2** Resumen de transformaciones

Transformación	Detalles	Cuándo utilizarla
Atributo a tipo de entidad	Reemplazar un atributo con un tipo de entidad y una relación 1-M.	Se necesitan detalles adicionales acerca de un atributo.
Dividir un atributo compuesto	Reemplazar un atributo con una colección de atributos.	Estandarizar los datos en un atributo.
Ampliar un tipo de entidad	Añadir un nuevo tipo de entidad y una relación 1-M.	Agregar un nivel más fino de detalle acerca de una entidad.
Entidad débil a entidad fuerte	Eliminar los símbolos de dependencia identificable y posiblemente añadir una llave primaria.	Eliminar las llaves foráneas combinadas después de la conversión en tablas.
Añadir historia	Para el atributo historia, reemplazar un atributo con un tipo de entidad y una relación 1-M. Para la relación historia, modificar la cardinalidad de la relación a M-N con un atributo. Para una historia limitada debe añadir atributos al tipo de entidad.	Añadir detalle para requerimientos legales o reportes estratégicos.
Añadir jerarquía de generalización	Comenzando con un supertipo: añada los subtipos, una jerarquía de generalización y redistribuya los atributos en subtipos. Comenzando con los subtipos: añada un supertipo, una jerarquía de generalización y redistribuya los atributos comunes y relaciones en el supertipo.	Clasificación aceptada de las entidades; atributos especializados y relaciones para los subtipos.

*TaxPayerID* y *EnterpriseZone*, mientras que los clientes comerciales tendrían valores nulos para *Subsidized* y *DwellingType*.

Esta transformación también se puede aplicar a una colección de tipos de entidad. En esta situación, la transformación involucra la adición de un supertipo y una jerarquía de generalización. Además, los atributos comunes en la colección de los tipos de entidad se mueven hacia el supertipo.

### 6.2.7 Resumen de transformaciones

Cuando se diseña una base de datos debe explorar de forma cuidadosa otros diseños. Las transformaciones descritas en esta sección le pueden ayudar a considerar otros diseños. Las transformaciones posibles no están limitadas a las descritas en esta sección. Puede invertir la mayoría de estas transformaciones. Por ejemplo, puede eliminar una jerarquía de generalización si los subtipos no tienen valores únicos. Para transformaciones adicionales debe revisar las referencias al final de este capítulo para consultar libros especializados de diseño de bases de datos. La tabla 6.2 presenta un resumen que le ayudará a recordar las transformaciones mostradas en esta sección.

## 6.3 Finalización de un ERD

---

Después de evaluar de forma iterativa los ERD alternativos utilizando las transformaciones presentadas en la sección 6.2, está listo para terminar su modelo de datos. Su modelo de datos no está completo sin la documentación de diseño adecuada y las cuidadosas consideraciones de los errores de diseño. Durante el proceso de diseño debe procurar escribir la documentación y llevar a cabo la revisión de errores del diseño. Incluso con la diligencia que amerita el proceso de diseño, necesitará realizar revisiones finales para asegurar una documentación de diseño adecuada y la carencia de errores en el diseño. Por lo general, estas revisiones se llevan a cabo con un equipo de diseñadores para asegurar la integridad. Esta sección presenta las guías para ayudarle a escribir la documentación del diseño y a revisar los errores del diseño.

### 6.3.1 Documentación de un ERD

El capítulo 5 (sección 5.4.1) prescribe documentación informal para las reglas de negocios que incluyen la integridad de los atributos, las restricciones de valores de los atributos, valores nulos y valores por omisión. Es importante documentar este tipo de reglas de negocios porque se pueden convertir en una especificación formal de SQL, como se describe en los capítulos 11 y 14. Para registrar este tipo de reglas de negocios debe utilizar la documentación informal asociada con los tipos de entidad, atributos y relaciones.

#### *Solución de problemas de especificación*

Más allá de la representación informal de las reglas de negocios, la documentación juega un papel importante en la solución de preguntas acerca de la especificación y para comunicar el diseño a otros. En el proceso de revisión de un ERD, debe documentar cuidadosamente las inconsistencias y falta de integridad en una especificación. Por lo general, una especificación grande contiene muchos puntos de inconsistencia y falta de integridad. El registro de cada punto permite la solución sistemática a través de actividades adicionales de obtención de requerimientos.

Ejemplifiquemos la inconsistencia: los requerimientos de la compañía de agua serían inconsistentes si una parte indica que un medidor está asociado con un cliente, pero otra parte establece que un medidor se puede asociar con múltiples clientes. Para resolver una inconsistencia un usuario puede indicar que la inconsistencia es una excepción. En este ejemplo, un usuario puede indicar las circunstancias en las cuales un medidor se puede asociar con varios clientes. El diseñador debe decidir sobre la solución del ERD, como permitir varios clientes para un medidor, permitir un segundo cliente o prohibir más de uno. El diseñador debe documentar cuidadosamente la solución de cada inconsistencia, incluyendo una justificación para cada solución.

Ejemplifiquemos la falta de integridad: la narrativa no especifica la cardinalidad mínima de un medidor en la relación *Uses* de la figura 6.2. El diseñador debe obtener los requerimientos adicionales para solucionar la especificación incompleta. Las partes incompletas de una especificación son comunes para las relaciones, ya que la especificación completa incluye dos conjuntos de cardinalidades. Es fácil omitir una cardinalidad de relación en una especificación inicial.

#### *Mejorando la comunicación*

Además de identificar los problemas de una especificación, la documentación debe utilizarse para comunicar el diseño a otros. Las bases de datos pueden tener un tiempo de vida largo, debido a la economía de los sistemas de información. Un sistema de información puede atravesar un ciclo grande de mantenimiento y mejoras antes de que exista una justificación suficiente para rediseñar el sistema. La buena documentación mejora un ERD, al comunicar la justificación para importantes decisiones del diseño. Su documentación no debe repetir las restricciones de un ERD. Por ejemplo, no necesita documentar que un cliente puede usar muchos medidores, ya que el ERD contiene esta información.

### documentación del diseño

incluye la justificación para las decisiones del diseño que involucran varias opciones factibles y explicaciones de las alternativas sutiles del diseño. No use la documentación únicamente para repetir la información que ya se encuentra en el ERD. También debe proporcionar una descripción para cada atributo, en especial, cuando el nombre de un atributo no indique su propósito. Conforme se desarrolla un ERD, debe documentar en los requerimientos de falta de integridad e inconsistencia.

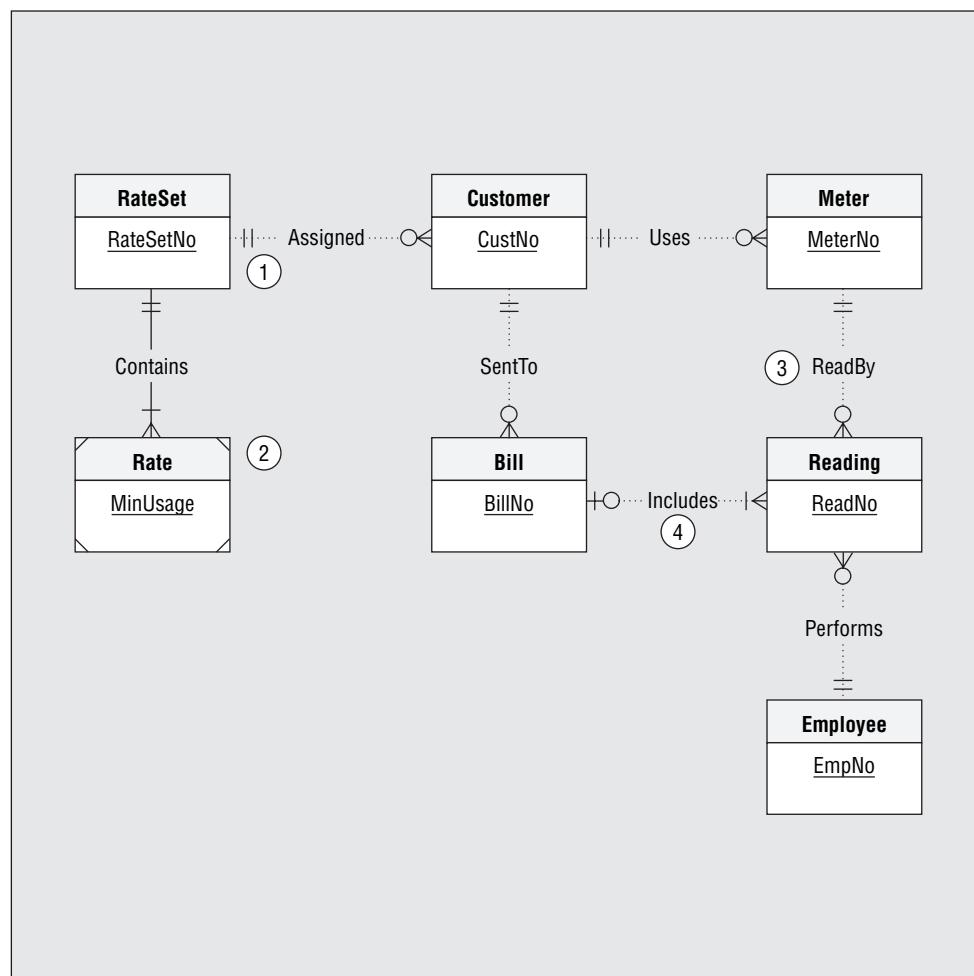
Debe documentar las decisiones en las que exista más de una opción factible. Por ejemplo, debe documentar cuidadosamente los diseños alternativos para las tarifas (un nivel de consumo individual *versus* niveles de consumo múltiples), como se ilustra en la figura 6.5. Debe documentar su decisión al registrar la recomendación y la justificación para la alternativa. Aunque todas las transformaciones presentadas en la sección previa pueden conducir a opciones factibles, debe enfocarse en las transformaciones más relevantes de la especificación.

También debe documentar las decisiones que puedan no ser claras para los demás aunque no existan alternativas factibles. Por ejemplo, la cardinalidad mínima de 0 del tipo de entidad *Reading* al tipo de entidad *Bill* quizás no sea clara. Debe documentar la necesidad de esta cardinalidad dada la diferencia de tiempo entre la creación de una factura y de sus lecturas asociadas. Un medidor se puede leer días antes de que se genere su factura asociada.

#### Ejemplo de documentación del diseño

La documentación del diseño debe incorporarse a su ERD. Si está usando una herramienta de dibujo que tenga un diccionario de datos, debe incluir las justificaciones del diseño en el diccionario de datos. La herramienta ER Assistant soporta las justificaciones de diseño así como los comentarios asociados con cada elemento de un diagrama. Puede usar los comentarios para describir el significado de los atributos. Si no está usando una herramienta que soporte la documentación, puede listar las justificaciones en una página separada y anotar su ERD, como se ilustra en la figura 6.10. Los números circunscritos de la figura 6.12 se refieren a las explicaciones de la tabla 6.3. Observe que algunos de los refinamientos mostrados previamente no se usaron en el ERD revisado.

**FIGURA 6.12**  
ERD de la compañía de agua revisado y con anotaciones



**TABLA 6.3**  
**Lista de justificaciones del diseño para el ERD revisado**

1. Un conjunto de tarifas es una colección de tarifas aprobadas por la comisión gubernamental de insumos.
2. Las tarifas son similares a las líneas de una tabla de impuestos. Se identifica una tarifa individual mediante el identificador de conjuntos de tarifas junto con el nivel del consumo mínimo de la tarifa.
3. La cardinalidad mínima indica que siempre se debe asociar un medidor con un cliente. Para una propiedad nueva, el desarrollador es inicialmente el responsable del medidor. Si el consumidor cancela una propiedad, la institución financiera que conserve la deuda será la responsable.
4. Una lectura no está asociada con una factura hasta que ésta se prepare. Se puede crear una lectura días antes de asociarle la factura.

**TABLA 6.4 Resumen de los errores de diseño**

Error de diseño	Descripción	Solución
Relación mal ubicada	Tipos de entidad conectados de forma incorrecta.	Considere todas las consultas que debe soportar la base de datos.
Relación faltante	Los tipos de entidad deben estar conectados de forma directa.	Examine las implicaciones de los requerimientos.
Cardinalidad incorrecta	Típicamente el uso de una relación 1-M en vez de una relación M-N.	Requerimientos incompletos: inferencias más allá de los requerimientos.
Abuso de las jerarquías de generalización	Las jerarquías de generalización no son comunes. Un error típico de los novatos es usarlas de forma inapropiada.	Asegúrese de que los subtipos tengan atributos y relaciones especializadas.
Abuso del tipo de entidad asociativo M-way	Las relaciones M-way no son comunes. Un error típico de los novatos es usarlas de forma inapropiada.	Asegúrese de que la base de datos registre las combinaciones de tres o más entidades.
Relación redundante	Relaciones derivadas de otras relaciones.	Examine cada ciclo de relaciones para revisar si una relación se puede derivar a partir de otras relaciones.

### 6.3.2 Detección de errores comunes de diseño

Como se indica en el capítulo 5, debe usar las reglas de diagramación para cerciorarse de que no existan errores obvios en su ERD. También debe usar las guías de esta sección para revisar si existen errores comunes del diseño. Los errores del diseño son más difíciles de solucionar que los errores del diagrama, debido a que involucran el significado de los elementos de un diagrama, y no sólo su estructura. Las siguientes subsecciones explican problemas comunes de diseño, mismos que se resumen en la tabla 6.4.

#### *Relaciones mal ubicadas o faltantes*

En un ERD grande es fácil conectar los tipos de entidad erróneos u omitir una relación requerida. Puede conectar los tipos de entidad erróneos si no toma en cuenta todas las consultas que debe soportar una base de datos. Por ejemplo, en la figura 6.12, si *Customer* se conecta directamente con *Reading* en lugar de conectarlo con *Meter*, no se puede establecer el control de un medidor, a menos que el medidor haya sido leído por el cliente. Las consultas que involucren el control de un medidor no podrán ser respondidas excepto a través de la lectura de los medidores.

Si los requerimientos no indican directamente una relación, debe considerar las implicaciones indirectas para detectar si se requiere una relación. Por ejemplo, los requerimientos para la base de datos de la compañía de agua no indican de forma directa la necesidad de una relación de *Bill* a *Reading*. Sin embargo, el cálculo cuidadoso del consumo revela la necesidad de una relación. La relación *Includes* conecta una factura con sus lecturas más recientes, por lo tanto, respalda el cálculo del consumo.

### *Cardinalidades incorrectas*

El error típico incluye el uso de una relación 1-M en lugar de una relación M-N. Este error pudo ser ocasionado por una omisión dentro de los requerimientos. Por ejemplo, si los requerimientos sólo indican que las asignaciones de trabajo involucran a una colección de empleados, usted no debe asumir que un empleado se puede relacionar con un sólo trabajo asignado. Debe obtener requerimientos adicionales para determinar si un empleado se puede asociar con varios trabajos asignados.

Otros errores de cardinalidad incorrecta que debe considerar son las cardinalidades inversas (la relación 1-M debe estar en dirección opuesta) y los errores en las cardinalidades mínimas. El error de la cardinalidad inversa típicamente se descuida. Las cardinalidades incorrectas indicadas en la especificación de las relaciones no son observables después de que se despliega el ERD. Debe revisar cuidadosamente todas las relaciones después de haber hecho la especificación para asegurar que haya consistencia. Los errores en la cardinalidad mínima típicamente son el resultado de omitir palabras clave en las narrativas de los problemas, tales como “opcional” y “requerido”.

### *Abuso de construcciones especializadas del modelado de datos*

Las jerarquías de generalización y los tipos de entidad asociativos M-way son construcciones especializadas del modelado de datos. Un error típico de los novatos es utilizarlas de forma inadecuada. No debe usar las jerarquías de generalización sólo porque una entidad puede existir con varios estados. Por ejemplo, el requerimiento de que la tarea de un proyecto puede comenzar, estar en proceso o terminarse no señala la necesidad de una jerarquía de generalización. La jerarquía de generalización es una herramienta adecuada si existe una clasificación establecida y atributos y relaciones especializados para los subtipos.

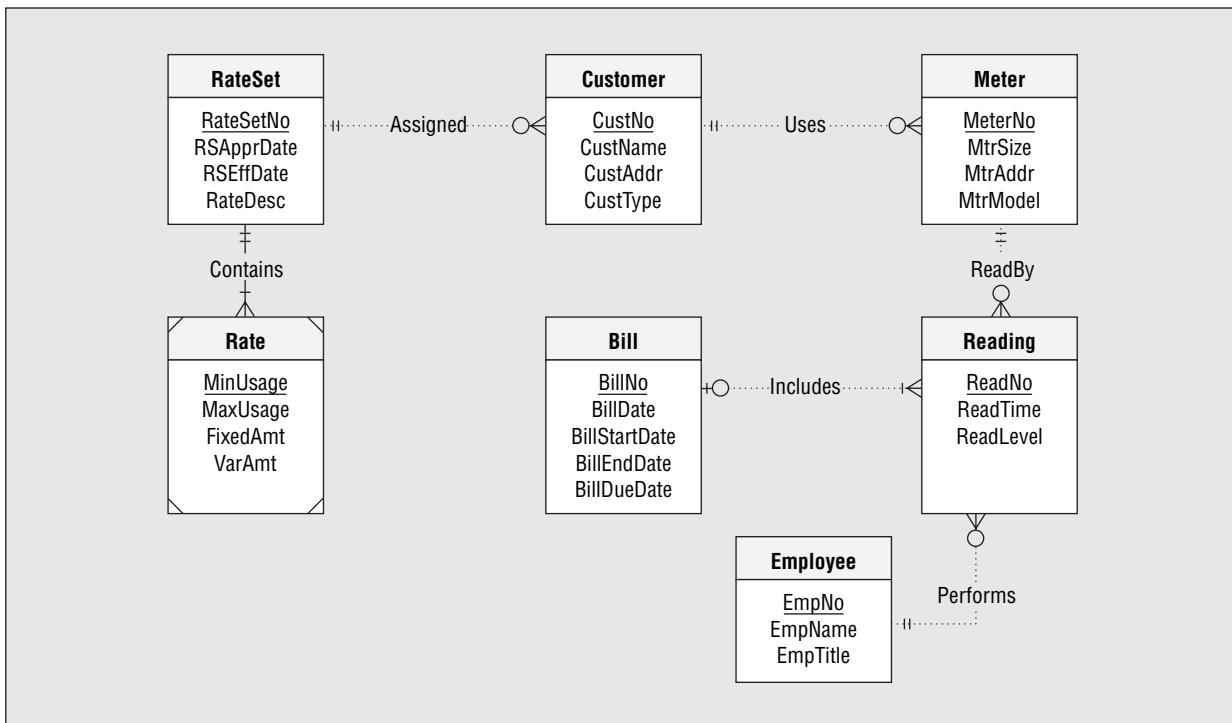
Un tipo de entidad asociativo M-way (un tipo de entidad asociativo que representa una relación M-way) se debe usar cuando la base de datos vaya a registrar combinaciones de tres (o más) objetos en lugar de combinaciones de dos objetos. En la mayoría de los casos sólo se deben registrar combinaciones de dos objetos. Por ejemplo, se deben usar relaciones binarias si una base de datos necesita registrar las habilidades de un empleado y las habilidades requeridas para un proyecto. Si una base de datos necesita registrar las habilidades de los empleados para proyectos específicos, se necesita un tipo de entidad asociativo M-way. Observe que la situación general con las relaciones binarias es más común que las situaciones posteriores representadas por un tipo de entidad asociativo M-way.

### *Relaciones redundantes*

Los ciclos en un ERD pueden señalar relaciones redundantes. Un ciclo incluye un conjunto de requerimientos acomodados de forma cíclica que comienza y termina con el mismo tipo de entidad. Por ejemplo, en la figura 6.10 existe un ciclo de relaciones que conectan *Customer*, *Bill*, *Reading* y *Meter*. Una relación es redundante en un ciclo si se puede obtener a partir de otras relaciones. Para la relación *SentTo*, las facturas asociadas con un cliente se pueden deducir de las relaciones *Uses*, *ReadBy* e *Includes*. En dirección opuesta, el cliente asociado con una factura se puede deducir a partir de las relaciones *Includes*, *ReadBy* y *Uses*. Aunque se puede asociar una factura con un conjunto de lecturas, cada lectura asociada se debe asociar con el mismo cliente. Debido a que la relación *SentTo* se puede deducir, se elimina en el ERD final (véase figura 6.13).

Otro ejemplo de relación redundante sería la relación entre *Meter* y *Bill*. Los medidores asociados con una factura se pueden deducir utilizando las relaciones *Includes* y *ReadBy*. Observe que el uso de grupos de tipos de entidad como *Reading* en el centro conectada con *Meter*, *Employee* y *Bill*, evitan las relaciones redundantes.

Debe tener cuidado cuando elimine las relaciones redundantes, ya que eliminar una relación necesaria es un error más grave que conservar una relación redundante. Cuando tenga duda, debe conservar la relación.

**FIGURA 6.13** ERD final de la compañía de agua

## 6.4 Conversión de un ERD en tablas relacionales

La conversión de la notación del ERD en tablas relacionales es importante dadas las prácticas de la industria. Las herramientas de ingeniería de software asistidas por computadora (CASE) soportan varias notaciones para los ERD. Es una práctica común usar una herramienta CASE como ayuda en el desarrollo del ERD. Debido a que la mayoría de los DBMS comerciales utilizan el modelo relacional, usted debe convertir un ERD en tablas relacionales para implementar su diseño de base de datos.

Incluso si usa una herramienta CASE para realizar la conversión, debe tener un entendimiento básico del proceso de conversión. La comprensión de las reglas de conversión mejoran su entendimiento del modelo ER, particularmente, la diferencia entre el modelo entidad-relación y el modelo relacional. Algunos errores típicos que cometen los modeladores novatos se deben a la confusión entre los modelos. Por ejemplo, el uso de llaves foráneas en un ERD se debe a la confusión de la representación de las relaciones en los dos modelos.

Esta sección describe el proceso de conversión en dos partes. Primero, se describen las reglas básicas para convertir los tipos de entidad, relaciones y atributos. Segundo, se muestran las reglas especializadas para convertir las relaciones opcionales 1-M, jerarquías de generalización y relaciones 1-1. Las sentencias CREATE TABLE de esta sección cumplen con la sintaxis SQL:2003.

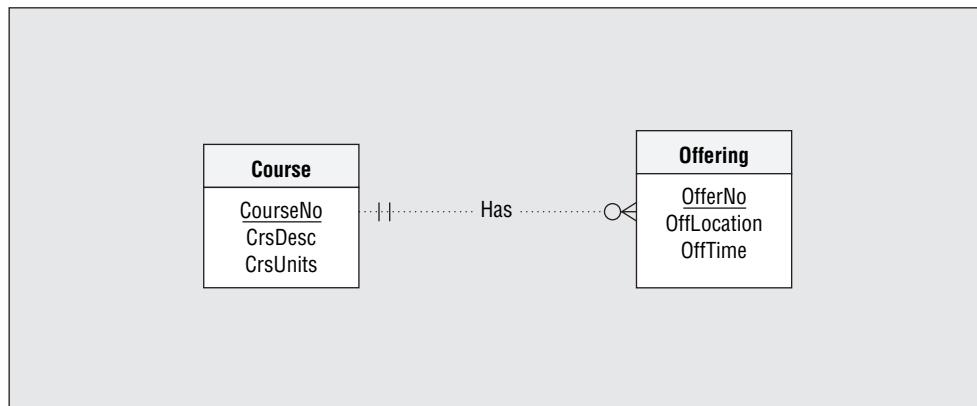
### 6.4.1 Reglas básicas de conversión

Las reglas básicas convierten todo lo que está en un ERD, excepto las jerarquías de generalización. Debe aplicar estas reglas hasta que todo lo que esté en su ERD sea convertido, a excepción de las jerarquías de generalización. Debe utilizar las primeras dos reglas antes que el resto de ellas. Conforme aplica estas reglas, puede usar una marca para señalar las partes convertidas del ERD.

1. **Regla del tipo de entidad:** Cada tipo de entidad (excepto los subtipos) se convierte en una tabla. La llave primaria del tipo de entidad (si no es débil) se convierte en la llave primaria de la tabla. Los atributos del tipo de entidad se convierten en las columnas de la tabla. Esta regla debe utilizarse primero, incluso antes de las reglas de relación.
2. **Regla de relación 1-M:** Cada relación 1-M se convierte en una llave foránea en la tabla que corresponda al tipo de entidad hija (el tipo de entidad cerca del símbolo de pata de cuervo). Si la cardinalidad mínima del lado madre de la relación es 1, la llave foránea no puede aceptar valores nulos.
3. **Regla de relación M-N:** Cada relación M-N se convierte en una tabla separada. La llave primaria de la tabla es una combinación de llaves formada por las llaves primarias de los tipos de entidad que participan en la relación M-N.
4. **Regla de dependencia identifiable:** Cada relación identifiable (representada por una línea de relación continua) agrega un componente a la llave primaria. La llave primaria de la tabla que corresponda a la entidad débil está formada por *i*) la llave local subrayada (si es que existe) de la entidad débil, y *ii*) la(s) llave(s) primaria(s) del(os) tipo(s) de entidad conectada por la relación identifiable.

Para comprender estas reglas, puede aplicarlas a algunos de los ERD utilizados en el capítulo 5. Utilizando las reglas 1 y 2 puede convertir la figura 6.14 en las sentencias CREATE TABLE que se muestran en la figura 6.15. La regla 1 se aplica para convertir los tipos de entidad

**FIGURA 6.14**  
ERD con una relación  
1-M



**FIGURA 6.15** Conversión de la figura 6.14 (sintaxis SQL:2003)

```

CREATE TABLE Course
(
    CourseNo CHAR(6),
    CrsDesc VARCHAR(30),
    CrsUnits SMALLINT,
    CONSTRAINT PKCourse PRIMARY KEY (CourseNo)
)

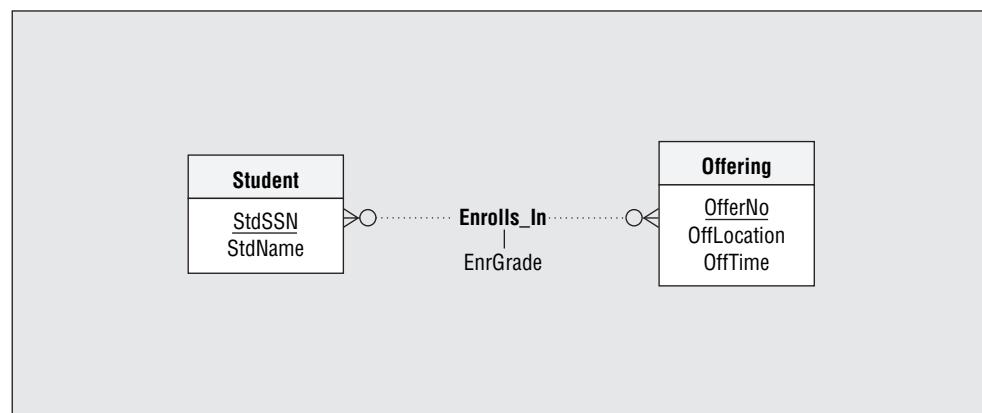
CREATE TABLE Offering
(
    OfferNo INTEGER,
    OffLocation CHAR(20),
    CourseNo CHAR(6) NOT NULL,
    OffTime TIMESTAMP,
    ...
    CONSTRAINT PKOffering PRIMARY KEY (OfferNo),
    CONSTRAINT FKCourseNo FOREIGN KEY (CourseNo) REFERENCES Course
)
  
```

*Course* y *Offering* en tablas. Después se aplica la regla 2 para convertir la relación *Has* en una llave foránea (*Offering.CourseNo*). La tabla *Offering* contiene la llave foránea, ya que el tipo de entidad *Offering* es el tipo de entidad hija en la relación *Has*.

Después, puede aplicar la regla de la relación M-N (regla 3) para convertir el ERD en la figura 6.16. Si sigue esta regla le conducirá a la tabla *Enrolls\_In* de la figura 6.17. La llave primaria de *Enrolls\_In* es una combinación de las llaves primarias de los tipos de entidad *Student* y *Offering*.

Para obtener práctica en la regla de dependencia identificable (regla 4), puede utilizarla para convertir el ERD de la figura 6.18. El resultado de convertir la figura 6.18 es idéntico a la figura 6.17, excepto que la tabla *Enrolls\_In* se renombra como *Enrollment*. El ERD de la figura 6.18 requiere dos aplicaciones de la regla de dependencia identificable. Cada aplicación de ésta agrega un componente a la llave primaria de la tabla *Enrollment*.

**FIGURA 6.16**  
Relación M-N con un atributo



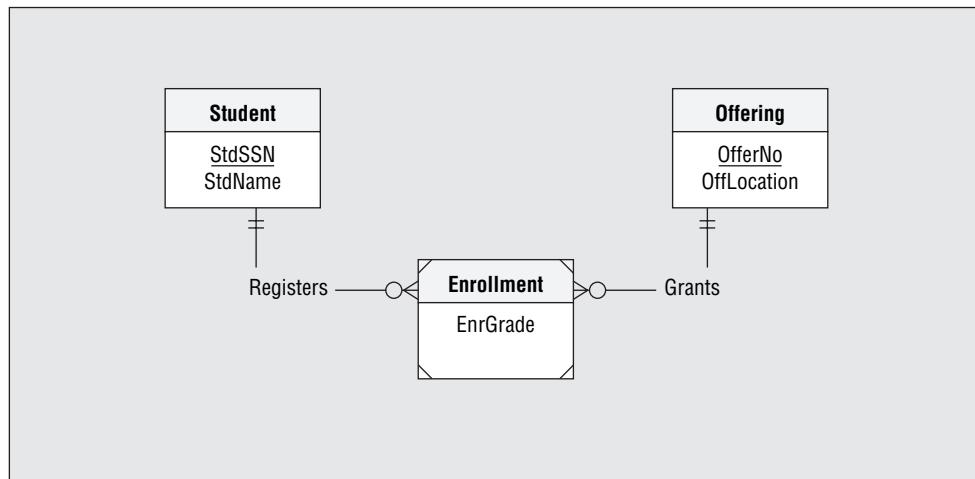
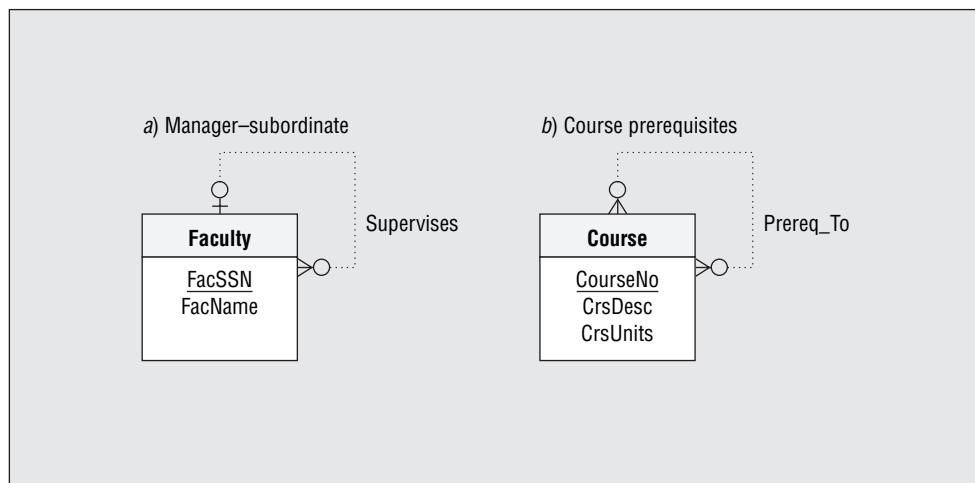
**FIGURA 6.17** Conversión de la figura 6.16 (sintaxis SQL:2003)

```

CREATE TABLE Student
(
    StdSSN           CHAR(11),
    StdName          VARCHAR(30),
    ...
    CONSTRAINT PKStudent PRIMARY KEY (StdSSN)
)

CREATE TABLE Offering
(
    OfferNo          INTEGER,
    OffLocation      VARCHAR(30),
    OffTime          TIMESTAMP,
    ...
    CONSTRAINT PKOffering PRIMARY KEY (OfferNo)
)

CREATE TABLE Enrolls_In
(
    OfferNo          INTEGER,
    StdSSN           CHAR(11),
    EnrGrade         DECIMAL(2,1),
    ...
    CONSTRAINT PKEnrolls_In PRIMARY KEY (OfferNo, StdSSN),
    CONSTRAINT FKOfferNo FOREIGN KEY (OfferNo) REFERENCES Offering,
    CONSTRAINT FKStdSSN FOREIGN KEY (StdSSN) REFERENCES Student
)
  
```

**FIGURA 6.18****Relación M-N*****Enrolls\_In* transformada en relaciones 1-M****FIGURA 6.19****Ejemplos de relaciones 1-M y M-N autorreferenciadas**

También puede aplicar las reglas para convertir las relaciones autorreferenciadas. Por ejemplo, puede aplicar las reglas de las relaciones 1-M y M-N para convertir las relaciones autorreferenciadas de la figura 6.19. Utilizando la regla de la relación 1-M, la relación *Supervises* se convierte en una llave foránea en la tabla *Faculty*, tal como se muestra en la figura 6.20. Usando la regla de la relación M-N, la relación *Prereq\_To* se convierte en la tabla *Prereq\_To* con una llave primaria combinada del número de curso del curso prerequisito y del número de curso del curso dependiente.

También puede aplicar las reglas de conversión a dependencias identificables más complejas, como se ilustra en la figura 6.21. La primera parte de la conversión es idéntica a la conversión de la figura 6.18. El uso de la regla 1-M hace que la combinación de *StdSSN* y *OfferNo* sean llaves foráneas de la tabla *Attendance* (figura 6.22). Observe que las llaves foráneas de *Attendance* hacen referencia a *Enrollment*, y no a *Sudent* y *Offering*. Finalmente, una aplicación de la regla de la dependencia identificable hace que la combinación de *StdSSN*, *OfferNo* y *AttDate* sean la llave primaria de la tabla *Attendance*.

La conversión de la figura 6.22 ilustra una situación en la cual se puede aplicar la transformación de una entidad débil en una fuerte (sección 6.2.3). En la conversión, la tabla *Attendance* contiene una llave foránea combinada (*OfferNo*, *StdSSN*). La modificación de *Enrollment* en una entidad fuerte eliminará la llave foránea combinada de la tabla *Attendance*.

FIGURA 6.20 Conversión de la figura 6.19 (sintaxis SQL:2003)

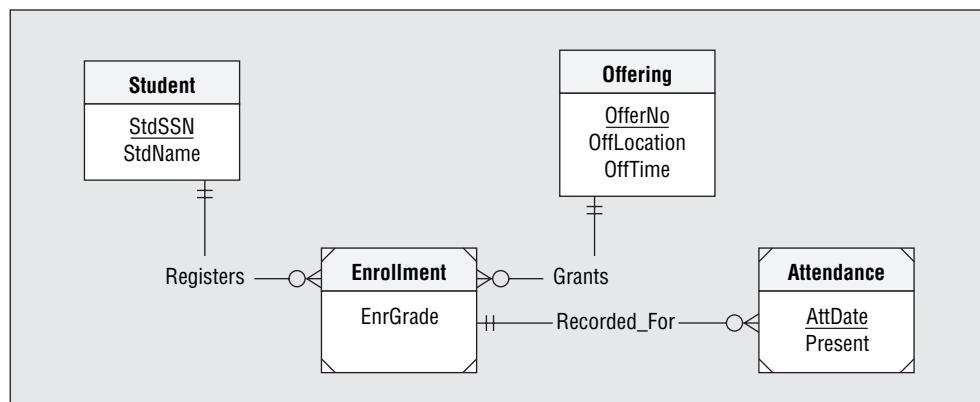
```

CREATE TABLE Faculty
(
    FacSSN           CHAR(11),
    FacName          VARCHAR(30),
    FacSupervisor    CHAR(11),
    ...
CONSTRAINT PKFaculty PRIMARY KEY (FacSSN),
CONSTRAINT FKSupervisor FOREIGN KEY (FacSupervisor) REFERENCES Faculty      )

CREATE TABLE Course
(
    Courseno         CHAR(6),
    CrsDesc          VARCHAR(30),
    CrsUnits         SMALLINT,
CONSTRAINT PKCourse PRIMARY KEY (CourseNo)      )

CREATE TABLE Prereq_To
(
    PrereqCNo        CHAR(6),
    DependCNo        CHAR(6),
CONSTRAINT PKPrereq_To PRIMARY KEY (PrereqCNo, DependCNo),
CONSTRAINT FKPrereqCNo FOREIGN KEY (PrereqCNo) REFERENCES Course,
CONSTRAINT FKDependCNo FOREIGN KEY (DependCNo) REFERENCES Course      )

```

FIGURA 6.21  
ERD con dos tipos de entidad débilFIGURA 6.22 Conversión del tipo de entidad *Attendance* de la figura 6.21 (sintaxis SQL:2003)

```

CREATE TABLE Attendance
(
    OfferNo          INTEGER,
    StdSSN           CHAR(11),
    AttDate          DATE,
    Present          BOOLEAN,
CONSTRAINT PKAttendance PRIMARY KEY (OfferNo, StdSSN, AttDate),
CONSTRAINT FKOfferNoStdSSN FOREIGN KEY (OfferNo, StdSSN)
    REFERENCES Enrollment      )

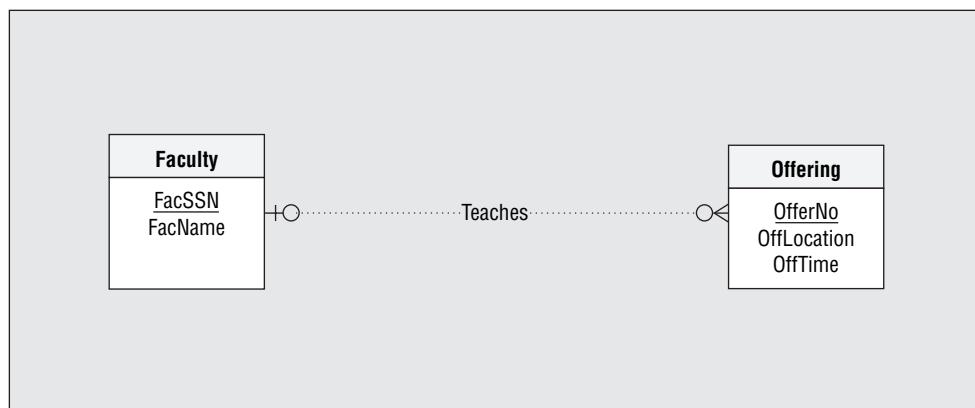
```

### 6.4.2 Conversión de relaciones opcionales 1-M

Cuando utiliza la regla de la relación 1-M para las relaciones opcionales, la llave foránea resultante contiene valores nulos. Recuerde que una relación con una cardinalidad mínima de 0 es opcional. Por ejemplo, la relación *Teaches* (figura 6.23) es opcional a *Offering*, ya que la entidad *Offering* se puede almacenar sin estar relacionada con la entidad *Faculty*. La conversión de la figura 6.23 genera dos tablas (*Faculty* y *Offering*), así como la llave foránea (*FacSSN*) de la tabla *Offering*. La llave foránea debe permitir valores nulos, ya que la cardinalidad mínima del tipo de entidad *Offering* en la relación es opcional (0). Sin embargo, los valores nulos pueden conducir a complicaciones en la evaluación de los resultados de las consultas.

Para evitar los valores nulos cuando se convierte una relación 1-M opcional, puede aplicar la regla 5 para convertir una relación 1-M opcional en una tabla, en lugar de una llave foránea. La figura 6.24 muestra una aplicación de la regla 5 al ERD de la figura 6.23. La tabla *Teaches* contiene las llaves foráneas *OfferNo* y *FacSSN* con valores nulos que no se permiten en ninguna de las columnas. Además, la tabla *Offering* ya no tiene la llave foránea que hace referencia a la tabla *Faculty*. Las figuras 6.25 y 6.26 ilustran un ejemplo de la conversión de una relación 1-M con un atributo. Observe que la tabla *Lists* contiene la columna *Commission*.

**FIGURA 6.23**  
Relación 1-M opcional



**FIGURA 6.24** Conversión de la figura 6.23 (sintaxis SQL:2003)

```

CREATE TABLE Faculty
(
    FacSSN           CHAR(11),
    FacName          VARCHAR(30),
    ...
CONSTRAINT PKFaculty PRIMARY KEY (FacSSN)  )

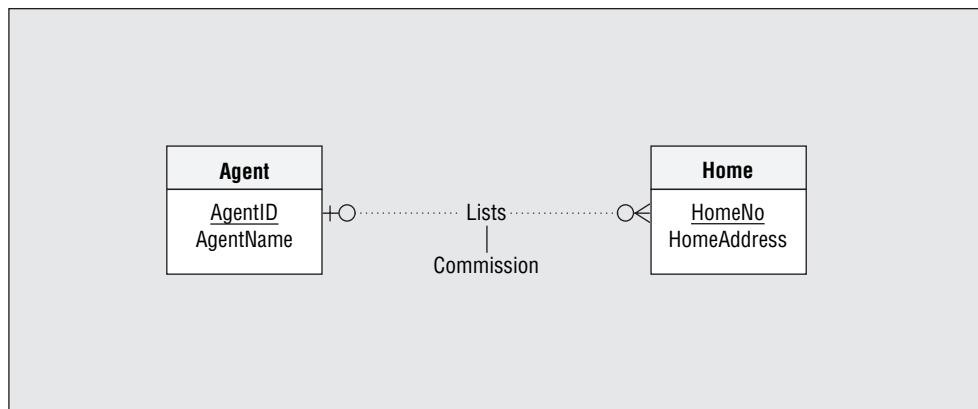
CREATE TABLE Offering
(
    OfferNo          INTEGER,
    OffLocation      VARCHAR(30),
    OffTime          TIMESTAMP,
    ...
CONSTRAINT PKOffering PRIMARY KEY (OfferNo)  )

CREATE TABLE Teaches
(
    OfferNo          INTEGER,
    FacSSN           CHAR(11)      NOT NULL,
    ...
CONSTRAINT PKTeaches PRIMARY KEY (OfferNo),
CONSTRAINT FKFacSSN FOREIGN KEY (FacSSN) REFERENCES Faculty,
CONSTRAINT FKOfferNo FOREIGN KEY (OfferNo) REFERENCES Offer  )
    
```

5. **Regla opcional de la relación 1-M:** Cada relación 1-M con la cardinalidad mínima de 0 en el lado de la madre se convierte en una tabla nueva. La llave primaria de la tabla nueva es la llave primaria del tipo de entidad del lado de la hija (muchos) de la relación. La tabla nueva contiene las llaves foráneas para las llaves primarias de ambos tipos de entidad que participan en la relación. Las dos llaves foráneas de la tabla nueva no permiten valores nulos. La tabla nueva también contiene los atributos opcionales de la relación 1-M.

La regla 5 es controvertida. El uso de la regla 5 en lugar de la regla 2 (regla de relación 1-M) evita que haya valores nulos en las llaves foráneas. Sin embargo, el uso de la regla 5 genera más tablas. La generación de las consultas puede ser más difícil con tablas adicionales. Además, la ejecución de las consultas puede ser más lenta debido a los enlaces adicionales. La selección de la regla 5 en lugar de la regla 2 depende de la importancia de la necesidad de evitar valores nulos contra la necesidad de evitar la generación de tablas adicionales. En muchas bases de datos, evitar la generación de tablas adicionales puede ser más importante que evitar los valores nulos.

**FIGURA 6.25**  
Relación 1-M opcional con un atributo



**FIGURA 6.26** Conversión de la figura 6.25 (sintaxis SQL:2003)

```

CREATE TABLE Agent
(
    AgentId           CHAR(10),
    AgentName         VARCHAR(30),
    ...
CONSTRAINT PKAgent PRIMARY KEY (AgentId) )

CREATE TABLE Home
(
    HomeNo            INTEGER,
    HomeAddress       VARCHAR(50),
    ...
CONSTRAINT PKHome PRIMARY KEY (HomeNo) )

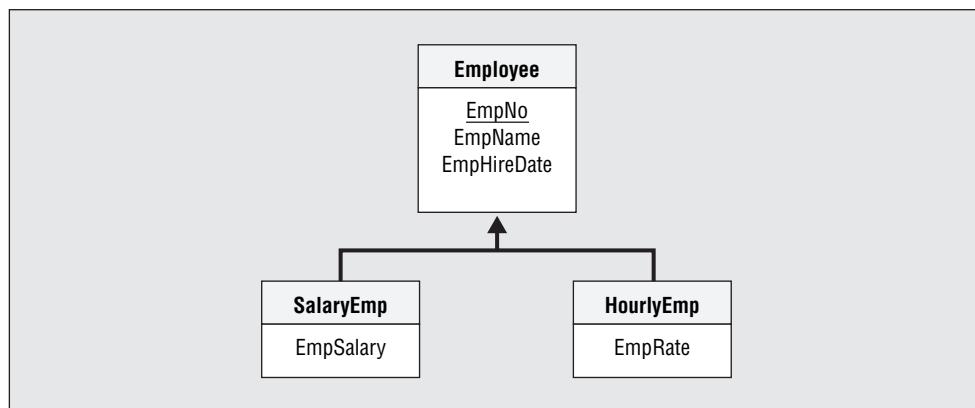
CREATE TABLE Lists
(
    HomeNo            INTEGER,
    AgentId          CHAR(10)      NOT NULL,
    Commission        DECIMAL(10,2),
    ...
CONSTRAINT PKLists PRIMARY KEY (HomeNo),
CONSTRAINT FKAgentId FOREIGN KEY (AgentId) REFERENCES Agent,
CONSTRAINT FKHomeNo FOREIGN KEY (HomeNo) REFERENCES Home )
    
```

### 6.4.3 Conversión de jerarquías de generalización

El alcance para convertir las jerarquías de generalización imita la notación entidad-relación tanto como es posible. La regla 6 convierte cada tipo de entidad de jerarquía de generalización en una tabla. La única columna que se muestra de una manera diferente a los atributos en la asociación del ERD es la llave primaria heredada. En la figura 6.27, *EmpNo* es una columna dentro de las tablas *SalaryEmp* y *HourlyEmp*, ya que es la llave primaria del tipo de entidad madre (*Employee*). Además, las tablas *SalaryEmp* y *HourlyEmp* tienen una restricción de llave foránea que apunta a la tabla *Employee*. La opción de eliminación CASCADE se configura en las dos restricciones de llaves foráneas (véase figura 6.28).

6. **Regla de jerarquía de generalización:** Cada tipo de entidad de una jerarquía de generalización se convierte en una tabla. Las columnas de una tabla son los atributos correspondientes a un tipo de entidad más la llave primaria del tipo de entidad madre. Para cada tabla que represente a un subtipo, se define una restricción de llave foránea que apunta a la tabla que corresponde a un tipo de entidad madre. Utilice la opción CASCADE para eliminar filas referenciadas.

**FIGURA 6.27**  
Jerarquía de generalización para los empleados



**FIGURA 6.28** Conversión de la jerarquía de generalización de la figura 6.27 (sintaxis SQL:2003)

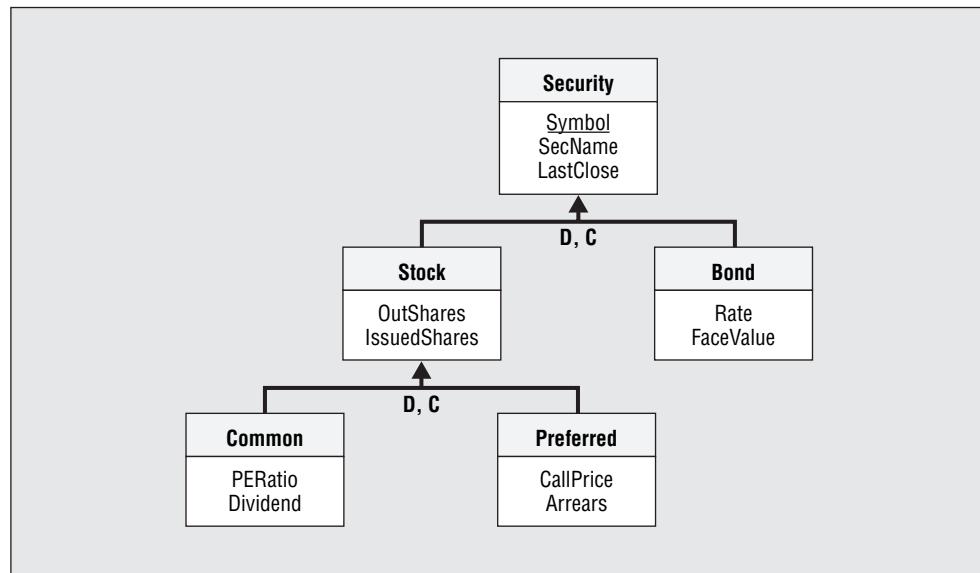
```

CREATE TABLE Employee
(
    EmpNo          INTEGER,
    EmpName        VARCHAR(30),
    EmpHireDate    DATE,
    CONSTRAINT PKEmployee PRIMARY KEY (EmpNo) )

CREATE TABLE SalaryEmp
(
    EmpNo          INTEGER,
    EmpSalary      DECIMAL(10,2),
    CONSTRAINT PKSalaryEmp PRIMARY KEY (EmpNo),
    CONSTRAINT FKSalaryEmp FOREIGN KEY (EmpNo) REFERENCES Employee
        ON DELETE CASCADE )

CREATE TABLE HourlyEmp
(
    EmpNo          INTEGER,
    EmpRate        DECIMAL(10,2),
    CONSTRAINT PKHourlyEmp PRIMARY KEY (EmpNo),
    CONSTRAINT FKHourlyEmp FOREIGN KEY (EmpNo) REFERENCES Employee
        ON DELETE CASCADE )
  
```

**FIGURA 6.29**  
Múltiples niveles de jerarquía de generalización



La regla 6 también se aplica a las jerarquías de generalización de más de un nivel. Para convertir la jerarquía de generalización de la figura 6.29 se generan cinco tablas (véase figura 6.30). En cada tabla se incluye la llave primaria madre (*Security*). Además, la restricción de la llave foránea se agrega a cada tabla correspondiendo a un subtipo.

Debido a que el modelo relacional no soporta directamente las jerarquías de generalización, existen varias maneras distintas para convertir las jerarquías de generalización. El otro alcance varía dependiendo del número de tablas y la ubicación de las columnas heredadas. La regla 6 puede generar enlaces adicionales o datos acerca de una entidad, pero no existen valores nulos y sólo pequeñas cantidades de datos duplicados. Por ejemplo, para concentrar todos los datos de una acción pública, debe enlazar las tablas *Common*, *Stock* y *Security*. Otros alcances acerca de las conversiones pueden requerir menos enlaces, pero generan datos más redundantes y valores nulos. Las referencias que se encuentran al final de este capítulo analizan las ventajas y desventajas de los distintos alcances para convertir las jerarquías de generalización.

También debe tomar en cuenta que las jerarquías de generalización para las tablas son soportadas en SQL:2003, el estándar emergente para las bases de datos relacionales de objetos que se presenta en el capítulo 18. En el estándar SQL:2003, las familias de subtablas proporcionan una forma de conversión directa desde las jerarquías de generalización, evitando la pérdida de información semántica cuando se realiza la conversión al modelo relacional tradicional. Sin embargo, pocos productos comerciales de DBMS soportan de manera completa las características de objetos relacionales en SQL:2003. Por lo tanto, probablemente sea necesario el uso de la regla de conversión de las jerarquías de generalización.

#### 6.4.4 Conversión de relaciones 1-1

A excepción de las jerarquías de generalización, las relaciones 1-1 no son comunes. Pueden ocurrir cuando las entidades con identificadores separados están muy relacionadas. Por ejemplo, la figura 6.31 muestra los tipos de entidad *Employee* y *Office* conectados por una relación 1-1. Los tipos de entidades que están separados parecen ser intuitivos, pero una relación 1-1 conecta los tipos de entidad. La regla 7 convierte las relaciones 1-1 en dos llaves foráneas, a menos de que se generen muchos valores nulos. En la figura 6.31, la mayoría de los empleados no administra oficinas. Por lo tanto, la conversión de la figura 6.32 elimina la llave foránea (*OfficeNo*) en la tabla de empleados.

7. **Regla de la relación 1-1:** Cada relación 1-1 se convierte en dos llaves foráneas. Si la relación es opcional con respecto a uno de los tipos de entidad, se puede eliminar la llave foránea correspondiente para deshacerse de los valores nulos.

FIGURA 6.30 Conversión de la jerarquía de generalización de la figura 6.29 (sintaxis SQL:2003)

```

CREATE TABLE Security
(
    Symbol           CHAR(6),
    SecName          VARCHAR(30),
    LastClose        DECIMAL(10,2),
CONSTRAINT PKSecurity PRIMARY KEY (Symbol) )

CREATE TABLE Stock
(
    Symbol           CHAR(6),
    OutShares        INTEGER,
    IssuedShares     INTEGER,
CONSTRAINT PKStock PRIMARY KEY (Symbol),
CONSTRAINT FKStock FOREIGN KEY (Symbol) REFERENCES Security ON DELETE CASCADE )

CREATE TABLE Bond
(
    Symbol           CHAR(6),
    Rate             DECIMAL(12,4),
    FaceValue        DECIMAL(10,2),
CONSTRAINT PKBond PRIMARY KEY (Symbol),
CONSTRAINT FKBond FOREIGN KEY (Symbol) REFERENCES Security ON DELETE CASCADE )

CREATE TABLE Common
(
    Symbol           CHAR(6),
    PERatio          DECIMAL(12,4),
    Dividend         DECIMAL(10,2),
CONSTRAINT PKCommon PRIMARY KEY (Symbol),
CONSTRAINT FKCommon FOREIGN KEY (Symbol) REFERENCES Stock ON DELETE
CASCADE )

CREATE TABLE Preferred
(
    Symbol           CHAR(6),
    CallPrice        DECIMAL(12,2),
    Arrears          DECIMAL(10,2),
CONSTRAINT PKPreferred PRIMARY KEY (Symbol),
CONSTRAINT FKPreferred FOREIGN KEY (Symbol) REFERENCES Stock
ON DELETE CASCADE )

```

FIGURA 6.31

Relación 1-1

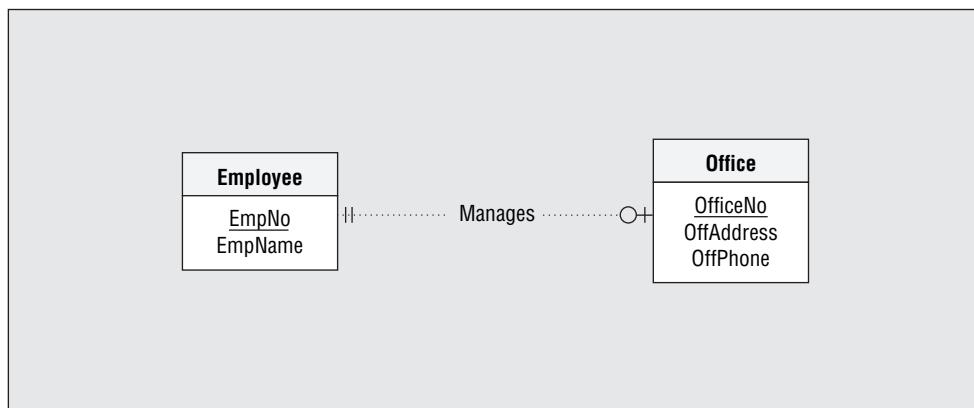


FIGURA 6.32 Conversión de la relación 1-1 de la figura 6.31 (sintaxis SQL:2003)

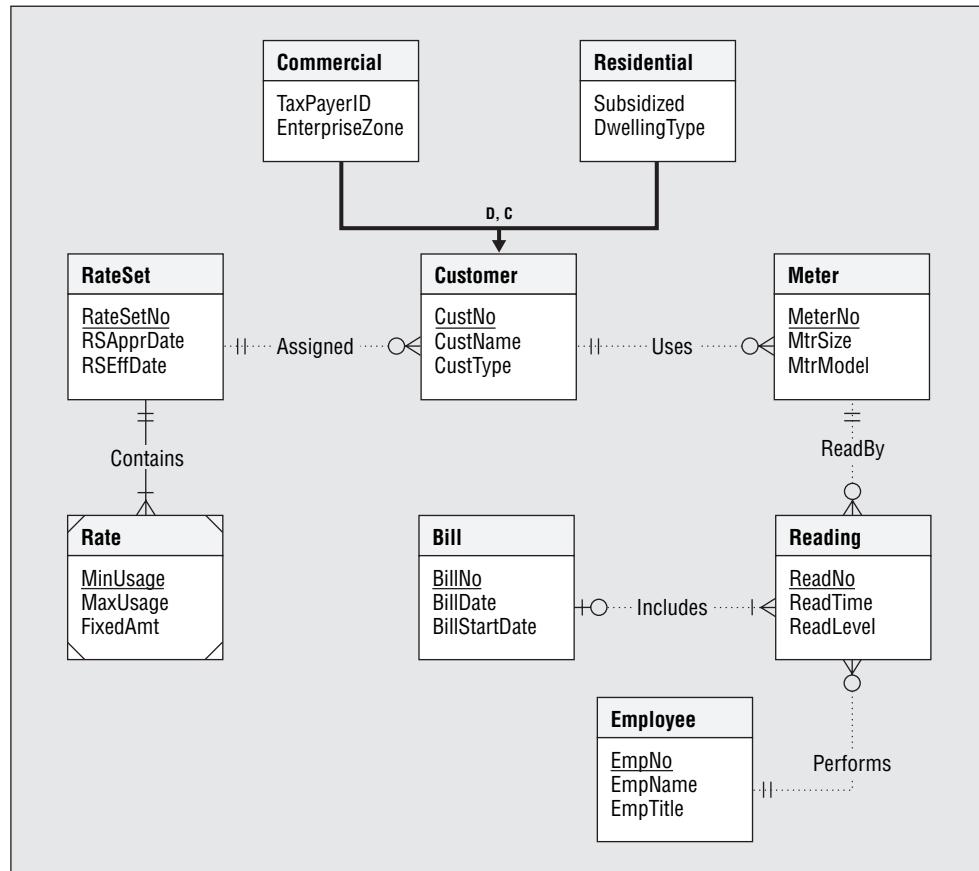
```

CREATE TABLE Employee
(
    EmpNo          INTEGER,
    EmpName        VARCHAR(30),
    CONSTRAINT PKEmployee PRIMARY KEY (EmpNo)
)

CREATE TABLE Office
(
    OfficeNo       INTEGER,
    OffAddress     VARCHAR(30),
    OffPhone       CHAR(10),
    EmpNo          INTEGER,
    CONSTRAINT PKOffice PRIMARY KEY (OfficeNo),
    CONSTRAINT FKEmpNo FOREIGN KEY (EmpNo) REFERENCES Employee,
    CONSTRAINT EmpNoUnique UNIQUE (EmpNo)
)

```

**FIGURA 6.33**  
ERD de la compañía  
de agua con una jerarquía de generalización



#### 6.4.5 Ejemplo de conversión comprensiva

Esta sección presenta un ejemplo más grande para integrar sus conocimientos acerca de las reglas de conversión. La figura 6.33 muestra un ERD similar al ERD final del problema de la compañía de agua comentado en la sección 6.1. Por brevedad, se han omitido algunos atributos. La figura 6.34 muestra las tablas relacionales derivadas a través de las reglas de conversión. La tabla 6.5 lista las reglas de conversión utilizadas junto con breves explicaciones.

**FIGURA 6.34** Conversión del ERD de la figura 6.33 (sintaxis SQL:2003)

```

CREATE TABLE Customer
(
    CustNo          INTEGER,
    CustName        VARCHAR(30),
    CustType        CHAR(6),
    RateSetNo       INTEGER      NOT NULL,
CONSTRAINT PKCustomer PRIMARY KEY (CustNo),
CONSTRAINT FKRateSetNo FOREIGN KEY (RateSetNo) REFERENCES RateSet  )

CREATE TABLE Commercial
(
    CustNo          INTEGER,
    TaxPayerID     CHAR(20)     NOT NULL,
    EnterpriseZone BOOLEAN,
CONSTRAINT PKCommercial PRIMARY KEY (CustNo),
CONSTRAINT FKCommercial FOREIGN KEY (CustNo) REFERENCES Customer
    ON DELETE CASCADE  )

CREATE TABLE Residential
(
    CustNo          INTEGER,
    Subsidized      BOOLEAN,
    DwellingType    CHAR(6),
CONSTRAINT PKResidential PRIMARY KEY (CustNo),
CONSTRAINT FKResidential FOREIGN KEY (CustNo) REFERENCES Customer
    ON DELETE CASCADE  )

CREATE TABLE RateSet
(
    RateSetNo       INTEGER,
    RSApprDate     DATE,
    RSEffDate      DATE,
CONSTRAINT PKRateSet PRIMARY KEY (RateSetNo)  )

CREATE TABLE Rate
(
    RateSetNo       INTEGER,
    MinUsage       INTEGER,
    MaxUsage       INTEGER,
    FixedAmt      DECIMAL(10,2),
CONSTRAINT PKRate PRIMARY KEY (RateSetNo, MinUsage),
CONSTRAINT FKRateSetNo2 FOREIGN KEY(RateSetNo) REFERENCES RateSet  )

CREATE TABLE Meter
(
    MeterNo         INTEGER,
    MtrSize         INTEGER,
    MtrModel        CHAR(6),
    CustNo          INTEGER      NOT NULL,
CONSTRAINT PKMeter PRIMARY KEY (MeterNo),
CONSTRAINT FKCustomer FOREIGN KEY (CustNo) REFERENCES Customer  )

```

**FIGURA 6.34 (Continuación)**

```

CREATE TABLE Reading
(
    ReadNo          INTEGER,
    ReadTime        TIMESTAMP,
    ReadLevel       INTEGER,
    MeterNo         INTEGER      NOT NULL,
    EmpNo           INTEGER      NOT NULL,
    BillNo          INTEGER,
)
CONSTRAINT PKReading PRIMARY KEY (ReadNo),
CONSTRAINT FKEmpNo FOREIGN KEY (EmpNo) REFERENCES Employee,
CONSTRAINT FKMeterNo FOREIGN KEY (MeterNo) REFERENCES Meter,
CONSTRAINT FKBillNo FOREIGN KEY (BillNo) REFERENCES Bill  )

CREATE TABLE Bill
(
    BillNo          INTEGER,
    BillDate        DATE,
    BillStartDate   DATE,
)
CONSTRAINT PKBill PRIMARY KEY (BillNo)  )

CREATE TABLE Employee
(
    EmpNo           INTEGER,
    EmpName         VARCHAR(50),
    EmpTitle        VARCHAR(20),
)
CONSTRAINT PKEmployee PRIMARY KEY (EmpNo)  )

```

**TABLA 6.5**  
**Reglas de conversión utilizadas para la figura 6.33**

Regla	Modo de empleo
1	Todos los tipos de entidad excepto los subtipos convertidos en tablas con llaves primarias.
2	Las relaciones 1-M convertidas en llaves foráneas: <i>Contains</i> se relaciona con <i>Rate</i> . <i>RateSetNo</i> ; <i>Uses</i> se relaciona con <i>Meter.CustNo</i> ; <i>ReadBy</i> se relaciona con <i>Reading.MeterNo</i> ; <i>Includes</i> se relaciona con <i>Reading.BillNo</i> ; <i>Performs</i> se relaciona con <i>Reading.EmpNo</i> ; <i>Assigned</i> se relaciona con <i>Customer.RateSetNo</i> .
3	No se usa ya que no existen relaciones M-N.
4	La llave primaria de la tabla <i>Rate</i> es una combinación de <i>RateSetNo</i> con <i>MinUsage</i> .
5	No se usa, aunque se puede utilizar para la relación <i>Includes</i> .
6	Subtipos ( <i>Commercial</i> y <i>Residential</i> ) convertidos en tablas. La llave primaria de <i>Customer</i> se agrega a las tablas <i>Commercial</i> y <i>Residential</i> . Las restricciones de llaves foráneas con las opciones CASCADE DELETE agregadas a las tablas correspondientes con los subtipos.

## Reflexión final

Este capítulo describió la práctica del modelado de datos a partir de la comprensión de la notación de pata de cuervo presentada en el capítulo 5. Para dominar el modelado de datos necesita comprender la notación utilizada en los diagramas de entidad-relación (ERD) y obtener mucha práctica en la construcción de ERD. Este capítulo describió las técnicas para deducir un ERD inicial a partir de la narrativa de un problema, refinar el ERD a través de transformaciones, documentar decisiones importantes del diseño y revisar el ERD para observar si existen errores de diseño. Para aplicar estas técnicas se presentó un problema práctico para una base de datos de la compañía de agua. Se le sugiere que aplique estas técnicas en los problemas al final del capítulo.

El resto de este capítulo presentó las reglas para convertir un ERD en tablas relacionales y en notaciones alternativas del ERD. Las reglas le ayudarán a convertir ERD de un tamaño modesto en tablas. Para problemas más grandes debe usar una buena herramienta CASE. Incluso si usa una herramienta CASE, la comprensión de las reglas de conversión le proporciona un vistazo sobre las diferencias del modelo entidad-relación y del modelo relacional.

Este capítulo se enfocó en las habilidades del modelado de datos para construir ERD utilizando las narrativas de problemas, refinación de ERD y conversión de ERD en tablas relacionales. El siguiente capítulo presenta la normalización, una técnica para eliminar la redundancia de las tablas relacionales. De forma conjunta, el modelado de datos y la normalización son habilidades fundamentales para el desarrollo de bases de datos.

Después de dominar estas habilidades de desarrollo de bases de datos, usted estará listo para aplicarlas a los proyectos de diseño de bases de datos. Un reto adicional para aplicar sus habilidades es la definición de requerimientos. Es muy laboriosa la recopilación de requerimientos de los usuarios con intereses y antecedentes diversos. Es posible que pase mucho tiempo obteniendo requerimientos y realizando el modelo de datos y la normalización. Con un estudio cuidadoso y práctica, encontrará que el desarrollo de bases de datos es una actividad que presenta retos y es muy bien pagada.

## **Revisión de conceptos**

- Identificar tipos de entidad y atributos en una narrativa.
- Criterios para las llaves primarias: estables y de propósito único.
- Identificar las relaciones de una narrativa.
- Transformaciones para agregar detalle a un ERD: atributo al tipo de entidad, ampliación de un tipo de entidad, adición de historia.
- División de un atributo para estandarizar el contenido de información.
- Modificación de una entidad débil en una entidad fuerte para eliminar las llaves foráneas combinadas después de la conversión.
- Añadir una jerarquía de generalización para evitar valores nulos.
- Prácticas de documentación para las decisiones importantes del diseño: justificación de las decisiones del diseño que involucren varias opciones factibles y explicación de las opciones sútiles del diseño.
- Prácticas de documentación pobre: repetición de la información que ya contiene un ERD.
- Errores comunes de diseño: relaciones mal ubicadas, relaciones faltantes, cardinalidades incorrectas, abuso de las jerarquías de generalización, abuso de los tipos de entidad asociativos representando relaciones M-way, y relaciones redundantes.
- Reglas básicas para convertir los tipos de entidad y relaciones.
- Reglas de conversión especializadas para convertir las relaciones 1-M opcionales, jerarquías de generalización y relaciones 1-1.

## **Preguntas**

1. ¿Qué significa decir que la construcción de un ERD es un proceso iterativo?
2. ¿Por qué conviene descomponer un atributo compuesto en atributos más pequeños?
3. ¿Cuándo se considera apropiado transformar un atributo en un tipo de entidad?
4. ¿Por qué transformar un tipo de entidad en dos tipos de entidad y una relación?
5. ¿Por qué transformar una entidad débil en una fuerte?
6. ¿Por qué transformar un tipo de entidad en una jerarquía de generalización?
7. ¿Por qué se añade historial a un atributo o relación?
8. ¿Cuáles son los cambios necesarios en un ERD cuando se transforma un atributo en un tipo de entidad?

9. ¿Qué cambios se necesita hacer a un ERD cuando se divide un atributo compuesto?
10. ¿Qué cambios son necesarios en un ERD cuando se expande un tipo de entidad?
11. ¿Qué cambios son necesarios en un ERD cuando se transforma una entidad débil en una fuerte?
12. ¿Qué cambios son necesarios en un ERD cuando se añade historial a un atributo o relación?
13. ¿Qué cambios son necesarios en un ERD cuando se añade una jerarquía de generalización?
14. ¿Qué se debe documentar en un ERD?
15. ¿Qué se debe omitir en la documentación de un ERD?
16. ¿Por qué los errores de diseño son más difíciles de detectar y de resolver que los errores de diagramación?
17. ¿Qué es una relación mal ubicada y cómo se resuelve?
18. ¿Qué es una cardinalidad incorrecta y cómo se resuelve?
19. ¿Qué es una relación faltante y cómo se resuelve?
20. ¿Qué es el abuso de una jerarquía de generalización y cómo se resuelve?
21. ¿Qué es un ciclo de relación?
22. ¿Qué es una relación redundante y cómo se resuelve?
23. ¿De qué manera se convierte una relación M-N en un modelo relacional?
24. ¿Cómo se convierte una relación 1-M en un modelo relacional?
25. ¿Cuál es la diferencia entre la regla de relación 1-M y la regla de relación 1-M opcional?
26. ¿De qué manera se convierte un modelo de entidad débil en un modelo relacional?
27. ¿Cómo se convierte una jerarquía de generalización en un modelo relacional?
28. ¿Cómo se convierte una relación 1-1 en un modelo relacional?
29. ¿Cuáles son los criterios para elegir una llave primaria?
30. ¿Qué se debe hacer si la llave primaria propuesta no satisface el criterio?
31. ¿Por qué es necesario comprender el proceso de conversión, incluso cuando se emplea una herramienta CASE para desempeñar la conversión?
32. ¿Cuáles son las metas del análisis de un problema narrativo?
33. ¿Cuáles son algunas de las dificultades de los requisitos de recolección de datos para desarrollar un modelo de datos de negocios?
34. ¿Cómo se identifican los tipos de entidad en un problema narrativo?
35. ¿Cómo debe aplicarse el principio de simplicidad durante la búsqueda de tipos de entidad en un problema narrativo?
36. ¿Cómo se identifican las relaciones y cardinalidades en un problema narrativo?
37. ¿De qué forma se puede reducir el número de relaciones en un ERD inicial?

## Problemas

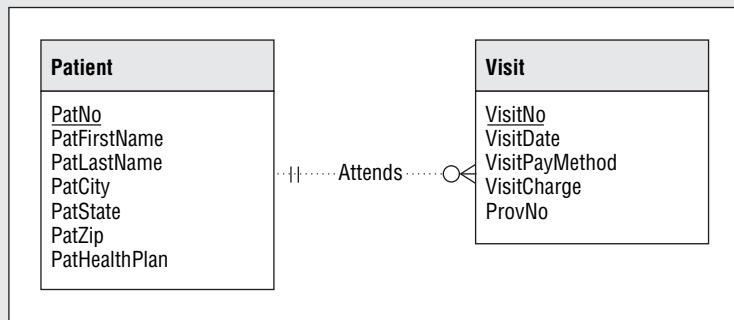
Los problemas se dividen en problemas de modelado de datos y problemas de conversión. En el capítulo 7 se encontrarán más problemas de conversión, donde ésta se presenta después de la normalización. Además de los problemas que se presentan aquí, el caso de estudio del capítulo 13 ofrece más oportunidades en la práctica de un problema más grande.

### Problemas de modelado de datos

1. Defina un ERD para la siguiente narrativa. La base de datos deberá rastrear hogares y propietarios. Una casa tiene un único identificador, dirección, ciudad, estado, código postal, número de recámaras, número de baños y pies cuadrados. Una casa está ocupada por su propietario o está rentada. Un propietario tiene un número de seguridad social, un nombre, el nombre de su cónyuge (opcional), una profesión y la profesión de su cónyuge (opcional). El propietario puede poseer una o más casas. Cada casa tiene sólo un dueño.
2. Refine el ERD del problema 1 añadiendo un tipo de entidad agente. Los agentes representan a los propietarios que venden una casa. Un agente puede enlistar muchas casas, pero sólo un agente puede enlistar una casa. Un agente posee un único identificador de agente, nombre, identificador de oficina y número telefónico. Cuando un propietario acuerda enlistar una casa con un agente, se determinan una comisión (porcentaje del precio de venta) y un precio de venta.
3. En el ERD del problema 2, transforme el atributo, identificador de oficina, en un tipo de entidad. Los datos de una oficina incluyen el número telefónico, el nombre del administrador y la dirección.

4. En el ERD del problema 3, añada el tipo de entidad comprador. Un tipo de entidad comprador tiene un número de seguridad social, un nombre, un número telefónico, preferencias por el número de recámaras y baños, y un rango de precio. Un agente puede trabajar con muchos compradores, pero un comprador trabaja con un solo agente.
5. Refine el ERD del problema 4 con una jerarquía de organización para destacar las similitudes entre compradores y propietarios.
6. Refine el ERD del problema 5 añadiendo un tipo de entidad oferta. Un comprador hace una oferta en una casa por un precio de venta específico. La oferta comienza con la fecha y la hora de presentación y expira en una fecha y hora determinadas. La oferta se identifica por un número único. Un comprador puede presentar múltiples ofertas por la misma casa.
7. Construya un ERD para representar cuentas en una base de datos para un software financiero personal. El software proporciona apoyo a cuentas de cheques, tarjetas de crédito y dos tipos de inversión (fondos mutuos y acciones). Ningún otro tipo de cuenta tiene apoyo, y cada una de ellas debe pertenecer a alguno de los tipos de cuenta que se han mencionado. Para cada clase de cuentas el software proporciona una pantalla separada de captura de datos. La siguiente lista describe los campos de las pantallas de captura de datos para cada clase de cuenta:
  - Para todas las cuentas, el software requiere de un único identificador de cuenta, nombre de la cuenta, fecha establecida y saldo.
  - Para las cuentas de cheques, el software soporta atributos para el nombre del banco, la dirección del mismo, el número de la cuenta de cheques y el número de enrutamiento.
  - Para las tarjetas de crédito, el software soporta atributos para el número de la tarjeta de crédito, la fecha de expiración y el límite de la tarjeta de crédito.
  - Para las acciones, el software soporta atributos para el símbolo y tipo de acción (común o preferente), la cantidad del último dividendo, la fecha del último dividendo, el tipo de cambio, el último precio de cierre, y el número de acciones (un número completo).
  - Para los fondos mutuos, el software soporta atributos para el símbolo del fondo mutuo, estado de las acciones (un número real), el tipo de fondo (acción, bono o mixto), el último precio de cierre, la región (nacional, internacional o global), y el estatus de exención de impuestos (sí o no).
8. Construya un ERD para representar las categorías de una base de datos para software financiero personal. Una categoría posee un único identificador de categoría, un nombre, un tipo (gasto, activo, pasivo o utilidad) y un saldo. Las categorías se organizan por jerarquías de manera que una categoría puede tener una categoría madre y una o más subcategorías. Por ejemplo, una categoría “propiedad” puede tener las subcategorías de “limpieza” o “mantenimiento”. Una categoría puede tener cualquier cantidad de niveles de subcategorías. Haga un diagrama de instancias para resaltar las relaciones que existen entre las categorías.
9. Diseñe un ERD para representar las partes y las relaciones entre ellas. Una parte tiene un único identificador, un nombre y un color. Una parte puede tener múltiples subpartes y múltiples partes que la usen. La cantidad de cada subparte debe ser registrada. Haga un diagrama de instancias para destacar las relaciones que hay entre las partes.
10. Diseñe un ERD para representar un estado de cuenta de una tarjeta de crédito. El estado consta de dos partes: un encabezado que contiene el número único del estado de cuenta, el número de cuenta del poseedor de la tarjeta de crédito y la fecha en la que se emite el estado de cuenta; y una sección detallada que contiene una lista de cero o más transacciones realizadas hasta la fecha de vencimiento del estado de cuenta. Cada línea de detalle contiene un número de línea, la fecha de la transacción, el nombre del comerciante y el monto de la transacción. El número de línea es único en el estado de cuenta.
11. Modifique su ERD para el problema 10. Todo es igual a excepción de que cada línea de detalle tiene un número único de transacción en lugar de un número de línea. Los números de transacciones son únicos en los estados de cuenta.
12. Por medio del uso del ERD de la figura 6.P1, transforme el atributo *ProvNo* en un tipo de entidad (*Provider*) y una relación 1-M (*Treats*). Un proveedor tiene un único número de proveedor, un primer nombre, un apellido, un número telefónico, una especialidad, el nombre del hospital en que el proveedor ejerce, una dirección de correo electrónico, una certificación, un grado de nómina y un título. Se requiere que el proveedor realice una visita, lo que no ocurre con proveedores nuevos, quienes no tienen visitas asociadas.
13. En el resultado del problema 12, expanda el tipo de entidad *Visit* para registrar los detalles de la misma. El detalle de la visita incluye número de detalle, cambio de detalle, número opcional de proveedor y un artículo asociado. La combinación del número de visita y el número del detalle de la visita es única para el detalle de la visita.

**FIGURA 6.P1**  
ERD para el problema 12



Un artículo incluye un número único de artículo, su descripción, su precio y el tipo del mismo. Un artículo puede estar relacionado con múltiples detalles de visita. Los nuevos artículos pueden no estar relacionados con ningún detalle de visita. Un proveedor puede estar relacionado con múltiples detalles de visita. Algunos proveedores pueden no estar asociados con detalles de visita alguno. Además, un proveedor puede relacionarse con múltiples visitas, como se indica en el problema 12.

14. En el resultado del problema 13, añada una jerarquía de generalización para distinguir entre proveedores de enfermería y médicos. Un enfermero tiene un grado de nómina y un título. Un médico tiene un hospital de residencia, una dirección de correo electrónico y una certificación. Los demás atributos del proveedor aplican a médicos y enfermeros. Una visita implica un proveedor médico, a la vez que un detalle de una visita puede involucrar a un proveedor de enfermería.
15. En el resultado del problema 14, transforme *VisitDetail* en una entidad fuerte con *VisitDetailNo* como la llave primaria.
16. En el resultado del problema 15, añada un historial para los precios de los artículos. Su solución debe dar soporte al precio actual junto con los dos precios más recientes. Incluya las fechas de los cambios para cada precio del artículo.
17. En el resultado del problema 15, añada un historial para los precios de los artículos. Su solución debe soportar ilimitado número de precios y fechas de cambios.
18. Diseñe un ERD con tipos de entidad para proyectos, especialidades y contratistas. Añada relaciones y/o tipos de entidad como se indica en la siguiente descripción. Cada uno de los contratistas tiene una sola especialidad, pero muchos contratistas pueden ofrecer la misma especialidad. Un contratista puede proveer la misma especialidad en varios proyectos. Un proyecto puede requerir de muchas especialidades y una especialidad puede emplearse en muchos proyectos. Cada combinación de proyecto y especialidad deberá tener por lo menos dos contratistas.
19. Para el siguiente problema, defina un ERD para los requisitos iniciales y después reviselo para los nuevos requisitos. Su solución deberá tener un ERD inicial, un ERD revisado y una lista de decisiones de diseño para cada ERD. Al llevar a cabo su análisis, usted quizá quiera seguir la metodología que se presenta en la sección 6.1. La base de datos apoya la oficina de colocaciones de una prestigiosa escuela de negocios. El propósito principal de la base de datos es programar entrevistas y facilitar las búsquedas por parte de los alumnos y las empresas. Considere los siguientes requisitos para su ERD inicial:
  - Los datos de los alumnos incluyen un identificador único de estudiante, un nombre, un número telefónico, una dirección de correo electrónico, una página web, una licenciatura, una especialidad y una calificación media.
  - La oficina de colocación mantiene una lista estándar de puestos basados en la lista de puestos del Departamento del Trabajo de Estados Unidos. Los datos del puesto incluyen un único identificador del puesto y la descripción del mismo.
  - Los datos de la compañía incluyen un identificador único de la compañía, el nombre y una lista de puestos y entrevistadores. Cada empresa debe colocar sus puestos en la lista de puestos que administra la oficina de colocación. Para cada puesto disponible, la empresa enlista las ciudades en las que dicho puesto está disponible.
  - Los datos del entrevistador incluyen un identificador único de entrevistador, un nombre, un número telefónico, una dirección de correo electrónico y una página Web. Cada entrevistador trabaja para una empresa.
  - Una entrevista abarca un único identificador de entrevista, una fecha, una hora, una ubicación (edificio y sala), un entrevistador y un alumno.

Después de revisar su diseño inicial, la oficina de colocación decide revisar los requerimientos. Haga un ERD separado para mostrar sus refinamientos. Refine el ERD original para dar apoyo a los siguientes nuevos requisitos:

- Permita a las empresas utilizar su propio lenguaje para describir los puestos. La oficina de colocación no mantendrá una lista para los puestos estándares.
- Permita que las empresas indiquen la disponibilidad de fechas y número de aperturas para posiciones.
- Permita que las empresas reserven bloques de horarios de entrevistas. Los bloques de entrevista no serán horas específicas para entrevistas en particular. En su lugar, una empresa solicitará bloques de X número de horas durante una semana en específico. Las compañías reservan bloques de entrevistas antes de que la oficina de colocación programe entrevistas individuales. De esta manera, la oficina de colocación necesita almacenar entrevistas además de bloques de entrevistas.
- Permita que los alumnos hagan ofertas para los bloques de entrevistas. Los alumnos reciben un monto establecido de dólares que pueden colocar entre sus ofertas. El mecanismo para presentar ofertas es una metodología de seudomercado que coloca entrevistas, un recurso escaso. Un alumno puede ofertar varias veces en un bloque de entrevistas y puede recibir a su vez, varias ofertas.

20. Para el siguiente problema, defina un ERD para los requisitos iniciales y después revise el ERD para los nuevos requisitos. Su solución deberá tener un ERD inicial, un ERD revisado y una lista de decisiones de diseño para cada ERD. Al realizar el análisis usted quizás quiera seguir la metodología presentada en la sección 6.1.

Diseñe una base de datos para administrar la asignación de tareas en una orden de trabajo. Una orden de trabajo registra la serie de tareas solicitadas por un cliente en una ubicación en particular.

- Un cliente tiene un único identificador de cliente, un nombre, una dirección para facturación (calle, ciudad, estado y código postal) y una colección de órdenes de trabajo que ha presentado.
- Una orden de trabajo tiene un número único de orden de trabajo, una fecha de creación, una fecha requerida, una fecha de término, un empleado supervisor opcional, una dirección de trabajo (calle, ciudad, estado y código postal) y una serie de tareas.
- Cada tarea tiene un único identificador de tarea, un nombre de tarea, una tarifa por hora y el estimado de horas. Las tareas están estandarizadas para todas las órdenes de trabajo, de manera que la misma tarea pueda desarrollarse en varias órdenes de trabajo.
- Cada una de las tareas en una orden de compra tiene un estado (sin iniciar, en progreso o completa), el número real de horas y una fecha de término. La fecha de término no se introduce hasta que el estado cambia a completo.

Después de revisar su diseño inicial, la compañía decide revisar los requisitos. Realice un ERD separado para mostrar sus refinamientos. Refine su ERD para dar apoyo a los siguientes requisitos:

- La compañía decide mantener una lista de materiales. Los datos acerca de los materiales incluyen un identificador único de material, un nombre y un costo estimado. Un material puede aparecer en varias órdenes de trabajo.
- Cada orden de trabajo tiene una colección de requerimientos de material. Un requerimiento de material incluye el material, una cantidad estimada del mismo y la cantidad real de material utilizado.
- El número estimado de horas para una tarea depende de la orden de trabajo y de la tarea, y no solamente de esta última. Cada tarea de una orden de trabajo incluye un número estimado de horas.

21. Para el siguiente problema defina un ERD para los requerimientos iniciales y después reviselo para los nuevos requerimientos. Su solución deberá contar con un ERD inicial, un ERD revisado y una lista de decisiones de diseño para cada uno de ellos. Para llevar a cabo su análisis quizás quiera seguir la metodología que se presenta en la sección 6.1.

Diseñe una base de datos que ayude al personal físico de la planta a administrar la asignación de llaves a los empleados. El propósito principal de la base de datos es asegurar el conteo preciso de todas las llaves.

- Un empleado cuenta con un número único de empleado, un nombre, un puesto y un número de oficina (opcional).
- Un edificio tiene un número único de edificio, un nombre y una ubicación dentro del campus.
- Una sala tiene un número de sala, un tamaño (dimensiones físicas), una capacidad, un número de entradas y la descripción del equipo de la sala. Debido a que cada sala está ubicada exactamente en un edificio, la identificación de la sala depende de la identificación del edificio.

- Las llaves tipo (también conocidas como llaves maestras) están diseñadas para abrir una o más salas. Una sala puede tener una o más llaves tipo que la abran. Una llave tipo tiene un número único de llave tipo, una fecha designada y el empleado que autoriza su uso. La creación de una llave tipo debe autorizarse con anterioridad.

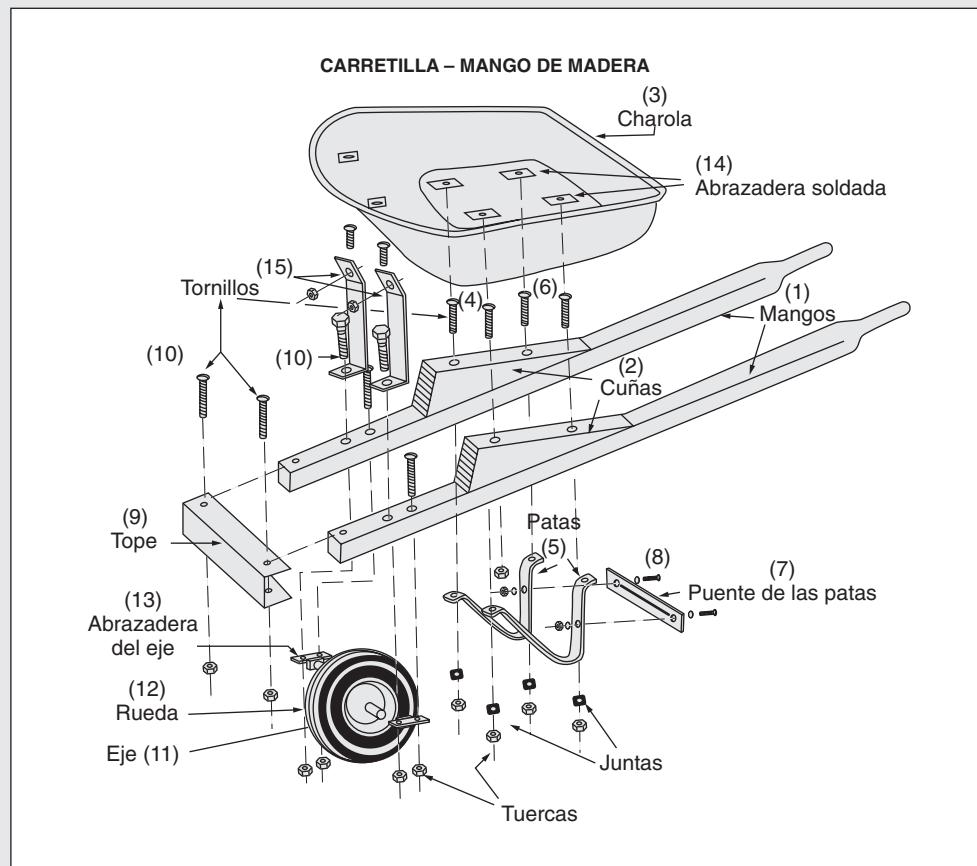
- A la copia de una llave tipo se le conoce como llave. Las llaves se asignan a los empleados. Cada llave se asigna exactamente a un empleado, pero un empleado puede tener varias llaves. El número de llave tipo más el número de la copia identifican de forma exclusiva a una llave. En la base de datos deberá registrarse la fecha en que se hizo la copia de una llave.

Después de revisar su diseño inicial, el supervisor físico de la planta decide revisar los requerimientos. Realice un ERD separado para mostrar sus refinamientos. Refine su ERD original para soportar los siguientes nuevos requisitos:

- La planta física necesita conocer no solamente al poseedor de la llave, sino a los antiguos poseedores de ésta. Para ellos, es necesario registrar el rango de fecha en que tuvieron control de la llave.
- La planta física requiere conocer el estado actual de cada llave: las que están en uso por parte de algún empleado, las que están almacenadas y las que se han reportado como perdidas. Si éste fuera el caso, se deberá registrar la fecha en la que se elaboró el reporte.

22. Defina un ERD que soporte la generación de diagramas de explosión de productos, instrucciones de ensamblaje y listado de partes. Estos documentos se incluyen por lo regular en productos que se venden al público. Su ERD debe representar los productos finales, además de las partes que los conforman. Los siguientes puntos proporcionan más detalles acerca de dichos documentos.
  - Su ERD deberá soportar la generación de los diagramas de explosión para una carretilla con mango de madera como se muestra en la figura 6.P2. Su ERD deberá almacenar las relaciones de contención junto con las cantidades requeridas para cada una de las subpartes. Para el dibujado de líneas y la posición geométrica de las especificaciones usted puede asumir que la imagen y los tipos de datos de posición están disponibles para almacenar los valores de los atributos.

**FIGURA 6.P2**  
Diagrama de explosión de producto



- Su ERD deberá soportar la generación de las instrucciones de ensamblado. Cada producto puede tener una serie de pasos ordenados para las instrucciones. La tabla 6.P1 indica algunas de las instrucciones de ensamblado para una carretilla. Los números en las instrucciones se refieren a las partes del diagrama.
- Su ERD deberá soportar la generación de un listado de partes para cada producto. La tabla 6.P2 muestra el listado de partes de la carretilla.



23. Para el ERD reporte de gastos que se muestra en la figura 6.P3, identifique y resuelva los errores y note lo que está incompleto en las especificaciones. Su solución deberá incluir una lista de errores y un ERD revisado. Para cada uno de los errores identifique el tipo de error (diagrama o diseño)

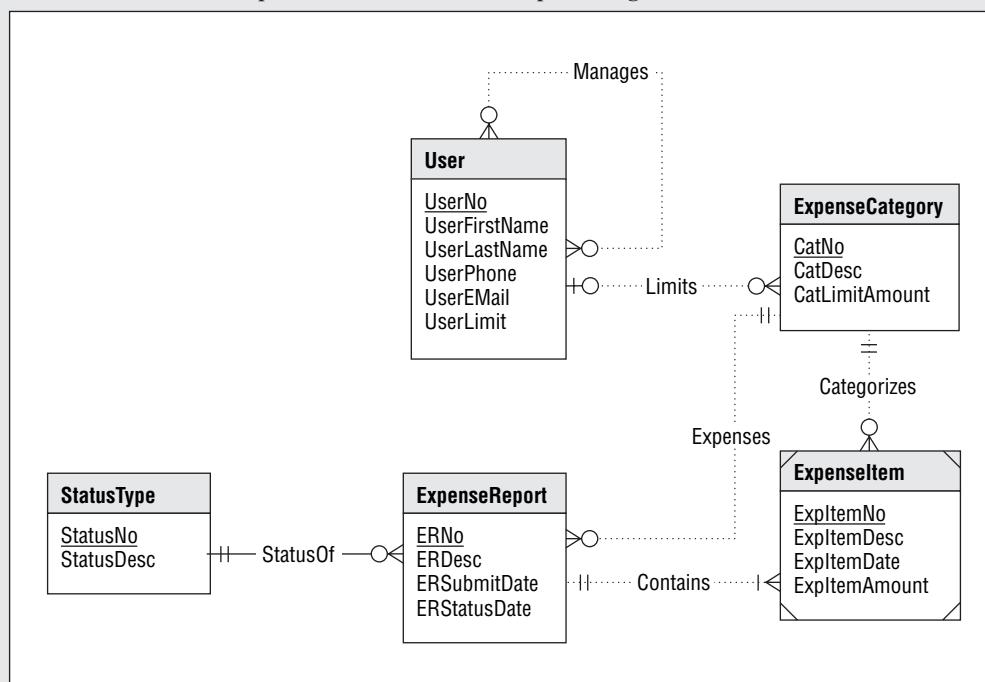
**TABLA 6.P1**  
**Ejemplo de instrucciones de ensamblado de la carretilla**

Paso	Instrucciones
1	El ensamblado requiere de pocas herramientas manuales, un destornillador, una llave de tuercas o "perico" para ajustar las tuercas.
2	NO apriete las tuercas con la llave hasta que haya armado la carretilla por completo.
3	Coloque los mangos (1) en dos cajas o en dos caballos de sierra (uno en cada extremo).
4	Coloque una cuña (2) encima de cada mango y alinee los orificios de los tornillos de la cuña con los orificios correspondientes en el mango.

**TABLA 6.P2**  
**Listado parcial de partes de la carretilla**

Cantidad	Descripción de la parte
1	Charola
2	Mango de madera
2	Cuña de madera
2	Pata

**FIGURA 6.P3** ERD para la base de datos del reporte de gastos



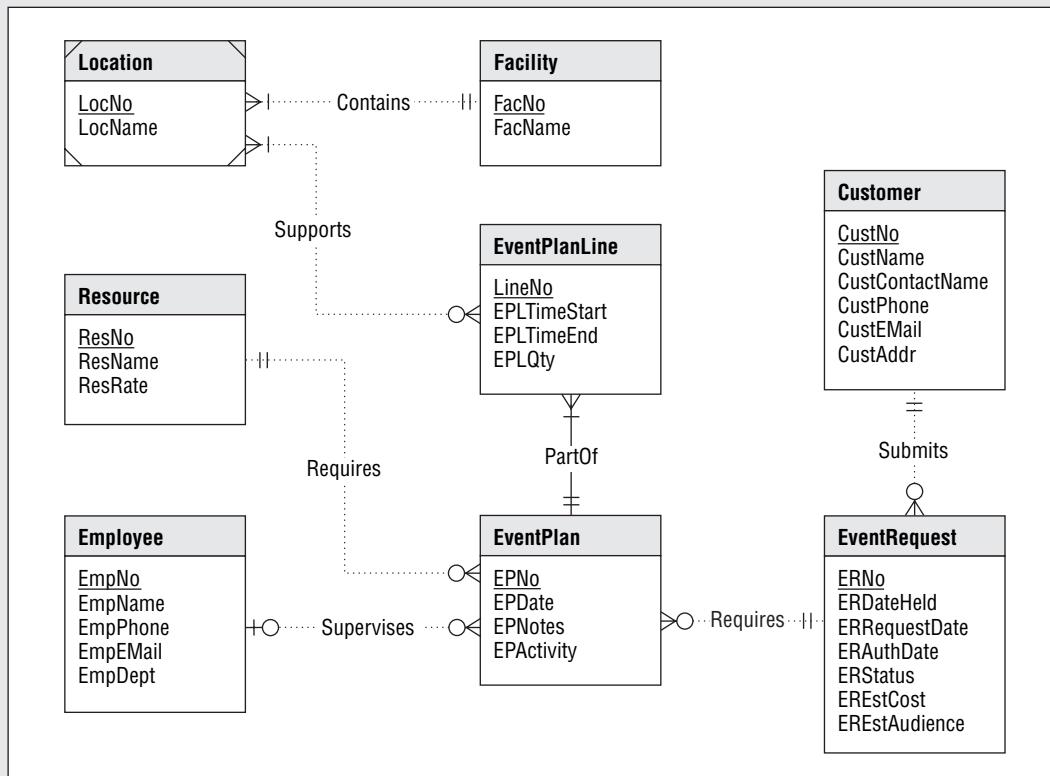
y el error específico dentro de cada tipo de error. Observe que el ERD podría tener errores en el diagrama y en el diseño. Si está utilizando ER Assistant, puede emplear la característica de Check Diagram después de revisar usted mismo las reglas del diagrama. Las especificaciones para el ERD se presentan a continuación:

- La base de datos reporte de gastos rastrea los reportes de gastos y los artículos del reporte de gastos, junto con los usuarios, categorías de gastos, códigos de estado y límites en la categoría de gasto realizado.
- Para cada usuario, la base de datos registra el número único de usuario, el nombre, el apellido, el número telefónico, la dirección de correo electrónico, el límite de gasto, las relaciones organizacionales entre los usuarios y las categorías de gastos disponibles para el usuario (por lo menos una). Un usuario puede administrar a otros usuarios, pero debe tener un sólo administrador como máximo. Existe un monto límite para cada categoría disponible para el usuario.
- Para cada categoría de gastos, la base de datos registra el número único de categoría, la descripción de la misma, el límite de gasto, y los usuarios autorizados a utilizar la categoría de gastos. Cuando se crea una categoría de gastos, es posible que no haya usuarios relacionados.
- Para cada código de estatus, la base de datos registra el número único del código de estado, la descripción del estatus y los informes de gastos por medio del código de estatus.
- Para cada informe de gastos, la base de datos registra el número único de informe de gastos, la descripción, la fecha en la que se presenta, la fecha del estatus, el código de estatus (requerido), el número de usuario (requerido) y los artículos de gasto relacionados.
- Para cada artículo de gasto, la base de datos registra el número único de gastos, la descripción, la fecha del gasto, el monto, la categoría del gasto (requerida) y el número de informe de gastos (requerido).



24. Para el ERD del Intercollegiate Athletic que se muestra en la figura 6.P4, identifique y resuelva los errores y observe lo que está incompleto en las especificaciones. Su solución debe incluir un listado de errores y un ERD revisado. Para cada uno de los errores, identifique el tipo de error (de diagramación o de diseño) y el error específico dentro de cada tipo de error. Observe que el ERD

**FIGURA 6.P4** ERD para la base de datos de Intercollegiate Athletic



podría tener errores en el diagrama y en el diseño. Si está utilizando ER Assistant, puede emplear la característica de Check Diagram después de revisar usted mismo las reglas del diagrama. Las especificaciones del ERD son las siguientes:

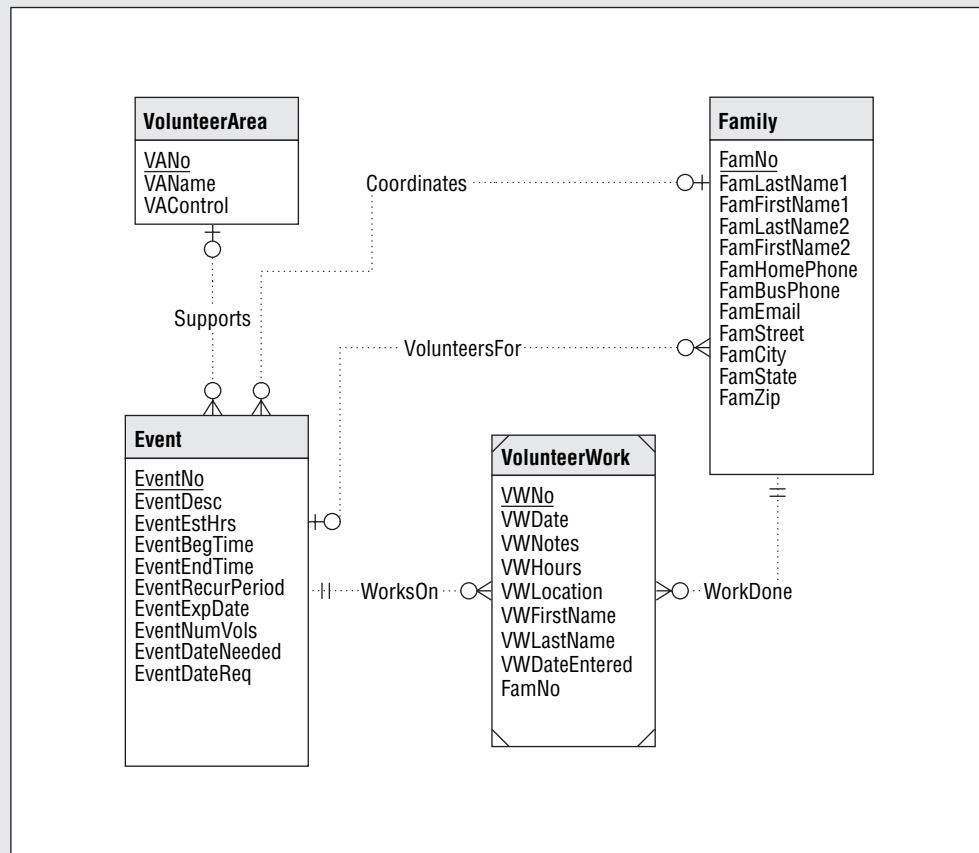
- La base de datos de Intercollegiate Athletic soporta la programación y operación de eventos, además de rastrear a los clientes, instalaciones, lugares con instalaciones, empleados y recursos para los eventos. Para programar un evento, el cliente inicia la solicitud con el departamento de Intercollegiate Athletic. Si se aprueba la solicitud, se hacen uno o más planes para el evento. De manera regular, los planes del evento constituyen el establecimiento, la operación y la limpieza del mismo. Un plan de un evento consiste en una o más líneas del plan de evento.
- Para cada solicitud de evento, la base de datos registra un número único de evento, la fecha en la que se lleva a cabo, la fecha solicitada, la fecha autorizada, el estado, un costo estimado, el público estimado, el número de instalación (requerido) y el número de cliente (requerido).
- Para cada plan de evento, la base de datos registra el número único de plan, las notas del plan, la fecha del trabajo, la actividad (establecimiento, operación o limpieza), el número de empleado (opcional) y el número de evento (requerido).
- Para cada plan de evento, la base de datos registra el número de línea (número único de plan), el número único de plan (requerido), la hora de inicio, la hora de finalización, el número de fuente (requerido), el número de ubicación (requerido) y la cantidad de recursos necesarios.
- Para cada cliente, la base de datos registra el número único de cliente, el nombre, la dirección, el nombre del contacto, el teléfono, la dirección de correo electrónico y la lista de eventos solicitados por el cliente. Un cliente no se almacena en la base de datos sino hasta que ha presentado una solicitud de evento.
- Para cada instalación, la base de datos registra el número único de instalación, el nombre de la misma y la lista de eventos en los que se solicita la instalación.
- Para cada empleado, la base de datos registra un número único de empleado, el nombre, el número de departamento, la dirección de correo electrónico, el número telefónico y la lista de planes de eventos supervisados por el empleado.
- Para cada ubicación, la base de datos registra el número relacionado de instalación, el número de la ubicación (único en una instalación), el nombre y la lista de líneas de planes de evento en los que se usa la ubicación.
- Para cada recurso, la base de datos registra el número único de registro, el nombre, la tarifa de la renta y la lista de líneas de plan de evento en las que se necesita el registro.



25. Para el ERD del Sistema de Información de Voluntarios que se muestra en la figura 6.P5, identifique y resuelva los errores y observe lo que está incompleto en las especificaciones. Su solución debe incluir un listado de errores y un ERD revisado. Para cada uno de los errores, identifique el tipo de error (de diagramación o de diseño) y el error específico dentro de cada tipo de error. Observe que el ERD podría tener errores en el diagrama y en el diseño. Si está utilizando ER Assistant, puede emplear la característica de Check Diagram después de revisar usted mismo las reglas del diagrama. Las especificaciones del ERD son las siguientes:

- El Sistema de Información de Voluntarios apoya a las organizaciones que necesitan rastrear voluntarios, áreas de voluntariado, eventos y número de horas trabajadas en un evento. Inicialmente el sistema será desarrollado para escuelas cuyos estatutos obligan a la participación de los padres de los alumnos como voluntarios. Los padres se registran como una familia de uno o dos padres. Los coordinadores de voluntarios los reclutan para las áreas de voluntarios. Los organizadores de eventos reclutan a los voluntarios para que trabajen en los eventos. Algunos eventos requieren de un horario del voluntario, mientras que otros no lo utilizan. Los voluntarios trabajan en los eventos y registran el tiempo que han trabajado.
- Para cada familia, la base de datos registra el número único de familia, el nombre y el apellido de cada uno de los padres, los teléfonos de casa y oficina, la dirección (calle, ciudad, estado y código postal) y una dirección opcional de correo electrónico. En el caso de las familias con un sólo parente, se registra solamente la información de un parente.
- Para cada área de voluntarios, la base de datos registra el número único de área de voluntarios, el nombre de la misma, el grupo que controla el área de voluntarios (facultad o asociación de padres de familia) y la familia que la coordina. En algunos casos, una familia podría coordinar más de un área de voluntarios.
- Para los eventos, la base de datos registra el número único de evento, la descripción del mismo, la fecha en que se lleva a cabo, la hora de inicio y término del evento, el número de voluntarios

**FIGURA 6.P5**  
ERD para el Sistema de Información de Voluntarios



requeridos, el periodo y la fecha de expiración del evento en caso de que se trate de un evento recurrente, el área de voluntarios y la lista de familias voluntarias para el evento. Las familias pueden programar con antelación participar como voluntarios en una serie de eventos que se presenten.

- Después de completar una asignación de trabajo se registran las horas trabajadas. La base de datos contiene el nombre y el apellido del voluntario, la familia a la que representa, el número de horas trabajadas, el evento opcional, la fecha trabajada, la ubicación del trabajo y comentarios opcionales. El evento es opcional para permitir que haya horas de trabajo voluntario que no sean consideradas como eventos.
- Defina un ERD que soporte la generación de guías de programación televisiva, listados de películas, deportes, acceso público y tablas de conversión de cable. Estos documentos se incluyen por lo regular en revistas sobre programas de televisión y periódicos dominicales. Además, estos documentos se encuentran disponibles en línea. Los siguientes puntos proporcionan más detalles sobre los documentos:
  - Una guía de programación televisiva enlista los programas disponibles durante cada fracción del día tal y como se ilustra en la figura 6.P6. Para cada programa en un canal/fracción de tiempo, la guía televisiva podría incluir todos o algunos de los siguientes atributos: título del programa, clasificación del contenido del programa, una descripción, un estado de repetición (sí o no), duración, estado de subtítulos (sí o no), y una hora de inicio si es que el programa no inicia en el lapso de 30 minutos. Para cada película, una guía también podría incluir todos o algunos de los siguientes atributos: una clasificación de evaluación (número de estrellas que va de 1 a 4, con incrementos de media estrella cada uno), un listado de los actores principales, una breve descripción (opcional), una clasificación del contenido de la película y el año de liberación de la misma. Los programas de acceso público se muestran en la guía de acceso público y no en la guía de programación televisiva.

**FIGURA 6.P6**  
Ejemplo de la guía de programación televisiva

CHANNELS	6 PM	6:30	7 PM	7:30
	CABLE CHANNELS CONTINUED			
68 68	Life Makeover Project		Sixteen Pepa's Fight 'TVPG'	
58 7	Ed McMahon's Next Big Star		Candid Camera	
61 61	Home Projects With Rick & Dan			
52 76	◀ Doctor Who ★ ★ ('96) 'TVPG'		The Addams Family ★ ★ ★	
25 25	Home Living - Lamps			
67 67	SoapTalk		Soapnet Special	
22 15 133	Bishop Jakes	Joyce Meyer	C. McClendon	Jack Hayford
57 6	◀ U.S. Marshals ★ ★ ('98, Crime drama) Tommy Lee Jones 'TV14'			
64 82	A Face in the Crowd ★ ★ ★ ↗ ('57) Andy Griffith, Patricia Neal			
44 44	Beyond Tough		Junkyard Wars	
47 47	⌚️◀ Arena Football (L)		Real TV	Real TV
51 51	◀ The Peacemaker ★ ★ ↗ ('97, Action) George Clooney 'TV14' (CC)			
59 78	America's Best Waterparks		America's Best Beaches 3	
66 86	Beaver	Beaver	Batman	Batman
33 33	The Rage: Carrie 2 ★ ↗ ('99) Emily Bergl, Jason London (CC)			
45 45	◀ Movie		Military Diaries	VH1 Special
69 69	(:15) Wall Street ★ ★ ★ ↗ ('87, Drama) Michael Douglas 'R'			
10 62	◀ Bull Durham ★ ★ ★ ('88)		Mutant X (R)	

- Un listado de películas incluye todas las películas que aparecen en la guía televisiva, como se muestra en la figura 6.P7. Para cada una de las películas el listado puede incluir alguno o todos estos atributos: un título, un año de liberación, una clasificación de evaluación, una clasificación del contenido, la abreviación del canal, un listado de combinación de los días de la semana/hora, un listado de actores principales, una breve descripción. Un listado de películas está organizado en orden ascendente según los títulos de las películas.
- Un listado de deportes contiene todos los eventos deportivos programados en la guía televisiva, como se muestra en la figura 6.P8. Un listado de deportes se organiza por deporte y día dentro de cada deporte. Cada punto en el listado de deporte puede incluir algunos o todos estos atributos: el título del evento, la hora, la duración, un canal, un indicador de subtítulo, un indicador de cuando el programa se presenta en vivo y un indicador de repetición.
- Un listado de acceso público muestra la programación de acceso público que no aparece en parte alguna en una guía de televisión, como se muestra en la figura 6.P9. Un listado de acceso público contiene un listado de organizaciones comunitarias (título, área, dirección por calle, ciudad,

FIGURA 6.P7 Ejemplo de listado de películas

MOVIES	
A	
<b>A.I.: ARTIFICIAL INTELLIGENCE</b> <i>Science fiction</i> in the Future, a cutting-edge android in the form of a boy embarks on a journey to discover its true nature. <i>Haley Joel Osment</i> (PG-13, 2:25) (AS, V) '01 ( <i>Esp.</i> ) iN1 June 2 3:30pm; 6 10:00am; 8 8:00am; 11 5:30pm; 13 10:00am; 25 10:00am; 29 9:00am (CC) , iN2 June 1 7:30pm; 8 6:00am; 19 6:30am; 11 3:30pm; 12 7:30am; 13 11:00am (CC) , iN3 June 5 9:00am, 11:30am, 2:00pm, 4:30pm, 7:00pm, 9:30pm (CC) , iN4 June 6 9:00am, 11:30am, 2:00pm, 4:30pm, 7:00pm, 9:30pm	<b>Power</b> (NR, 1:37) '57 ( <i>Esp.</i> ) TMAX June 4 6:05am; 21 4:40pm; 30 3:20pm
<b>A.K.A. CASSIUS CLAY</b> * * * <i>Documentary</i> Heavyweight boxing champ Muhammad Ali speaks, visits comic Stepin Fetchit and appears in fight footage. (PG, 1:25) (AS, L, V) '70 ( <i>Esp.</i> ) TMC June 1 6:15am; 6 2:30pm; 19 6:20am, TMC-W June 1 9:15am; 6 5:30pm; 10 9:20am	<b>ABBOTT AND COSTELLO MEET THE KILLER, BORIS KARLOFF.</b> * * * <i>Comedy</i> A hotel detective and bellhop find dead bodies and a fake swami. <i>Bud Abbott</i> (TVG, 1:30) '48 AMC June 28 5:30pm; 21 7:35am (CC)
<b>ABANDON SHIFT</b> * * * <i>Adventure</i> Short rations from a sunken liner force the officer of a packed lifeboat to sacrifice the weak. <i>Tyrone</i>	<b>ABBOTT AND COSTELLO MEET FRANKENSTEIN</b> * * * <i>Comedy</i> The Wolf Man tries to warn a dimwitted porter that Dracula wants his brain for a monster's body. <i>Bud Abbott</i> (TVG, 1:30) '48 AMC June 5 5:30pm (CC)
	<b>ABDUCTION OF INNOCENCE: A MOMENT OF TRUTH MOVIE</b> * * <i>Drama</i> A lumber magnate's teen daughter stands trial for being an accomplice in her own kidnapping. <i>Katie Wright</i> (TVPG, 1:45) '96 LMN June 1 8:00pm; 2 9:30am (CC)
	<b>THE ABDUCTION OF KARI SWENSON</b> * * <i>Docudrama</i> A U.S. Olympic biathlete is kidnapped in 1984 by father-and-son Montana mountain men. <i>Tracy Pollan</i> (TVPG, 1:45) (V) '87 LMN June 10 4:30pm; 11 6:00am
	<b>ABOUT ADAM</b> * * * <i>Romance-comedy</i> A magnetic young man meets and romances an Irish waitress, then courts and beds the rest of the family. <i>Stuart Townsend</i> (R, 1:45) (AS, L) '00 STARZIC June 22 8:00pm; 23 1:10pm; 27 2:30pm, 10:15pm (CC)
	<b>ABOUT SARAH</b> * * <i>Drama</i> A young woman decides whether to continue her medical career or return to her impoverished mother. <i>Kellie</i>
	<b>discovery.</b> <i>Ed Harris</i> (PG-13, 2:47) (AS, L, V) '89 ( <i>Esp.</i> ) ACTION June 2 12:05pm, 8:00pm, 3 6:45am; 13 12:20pm, 8:00pm; 22 8:10am, 5:35pm
	<b>THE ACCIDENT: A MOMENT OF THRUTH MOVIE</b> * * <i>Docudrama</i> A teen, charged with manslaughter in a drunken driving crash that killed her best friend, uses alcohol to cope. <i>Bonnie Root</i> (TVPG, 1:45) '97 LMN June 8 2:45pm (CC)
	<b>THE ACCIDENTAL TOURIST</b> * * * <i>Drama</i> A travel writer takes up with his dog trainer after his wife moves out. <i>William Hurt</i> (TVPG, 2:30) (AS, L) '88 FOX-WXIX June 23 12:00pm
	<b>THE ACCUSED</b> * * * <i>Crime drama</i> A psychology professor goes to trial for killing a student who tried to seduce her. <i>Loretta Young</i> (NR, 1:41) '48 TCM June 8 10:30am
	<b>AN ACT OF LOVE: THE PATRICIA NEAL STORY</b> * * * <i>Docudrama</i> The actress recovers from a 1966 stroke with help from friends and her husband, writer Roald Dahl. <i>Glenda Jackson</i> (NR, 1:40) '81 WE June 26 11:10am
	<b>ACTIVE STEALTH</b> <i>Action</i> When terrorists steal a stealth bomber, the Army calls upon a veteran fighter pilot and his squadron to retrieve it. <i>Daniel Baldwin</i> DC (R, 1:38) (AS, L, V) '99 ( <i>Esp.</i> ) AMAX June 2 2:45pm; 5 4:30pm; 7 8:00pm; 10 12:10pm; 15 6:20pm; 18 8:00pm; 24 12:50pm; 25 1:15pm; 30 1:15pm (CC)
	<b>THE ACTRESS</b> * * * <i>Drama</i> Supported by her mother, a New Englander finally tells her son...

estado, código postal y número telefónico). Después de enlistar las organizaciones comunitarias, un listado de acceso público contiene la programación para cada día/hora. Debido a que el acceso público no ocupa todas las secciones del día, hay una lista de las fracciones para cada día y no una cuadrícula, como sucede en el caso de la guía televisiva completa. Cada programa de acceso público tiene un título y una organización de comunidad de patrocinio opcional.

- Una tabla de conversión/cable muestra el mapeo de los canales a lo largo de los sistemas de cable, como se indica en la figura 6.P10. Para cada uno de los canales, la tabla de conversión muestra un número para cada sistema de cable en el área geográfica local.

FIGURA 6.P8 Ejemplo de listado de deportes

<b>Sports</b>	
8:00pm	<b>GOLF</b> Golf Murphy's Irish Open, First Round (R)
11:00pm	<b>GOLF</b> Golf Murphy's Irish Open, First Round (R)
<b>FRIDAY, JUNE 28</b>	
10:00am	<b>GOLF</b> Golf Murphy's Irish Open, Second Round (L)
12:00pm	<b>ESPN</b> U.S. Senior Open, Second Round (L) (CC)
2:00pm	<b>ESPN</b> PGA FedEx St. Jude Classic, Second Round (L) (CC)
3:00pm	<b>GOLF</b> ShopRite LPGA Classic, First Round (L)
4:00pm	<b>ESPN</b> Golf U.S. Senior Open, Second Round (L) (CC)
5:30pm	<b>GOLF</b> Scorecard Report
8:00pm	<b>GOLF</b> Golf ShopRite LPGA Classic, First Round (R)
10:00pm	<b>GOLF</b> Scoreboard Report
11:00pm	<b>GOLF</b> Golf Murphy's Irish Open, Second Round (R)
<b>SATURDAY, JUNE 29</b>	
10:00am	<b>GOLF</b> Golf Murphy's Irish Open, Third Round (L)
3:00pm	<b>NBC-WLWT</b> Golf U.S. Senior Open, Third Round (L) (CC)
4:00pm	<b>ABC-WCPO</b> PGA FedEx St. Jude Classic, Third Round (L)
4:30pm	<b>GOLF</b> ShopRite LPGA Classic, Second Round (L)
7:00pm	<b>GOLF</b> Scorecard Report
8:00pm	<b>GOLF</b> Haskins Award
8:30pm	<b>GOLF</b> Golf ShopRite LPGA Classic, Second Round (R)
10:00pm	<b>GOLF</b> Scorecard Report
11:00pm	<b>GOLF</b> Haskins Award
11:30PM	<b>GOLF</b> Golf Murphy's Irish Open, Third Round (R)
<b>HORSE EVENTS</b>	
<b>SUNDAY, JUNE 2</b>	
2:00pm	<b>ESPN</b> Equestrian Del Mar National (CC)
<b>WEDNESDAY, JUNE 5</b>	
2:00pm	<b>ESPN2</b> Wire to Wire
<b>FRIDAY, JUNE 7</b>	
5:00pm	<b>ESPN2</b> Horse Racing Acorn Stakes (L)
<b>SATURDAY, JUNE 8</b>	
2:00pm	<b>ESPN2</b> Horse Racing Belmont Stakes Special (L) (CC)
5:00pm	<b>NBC-WLWT</b> Horse Racing Belmont Stakes (L) (CC)
<b>WEDNESDAY, JUNE 12</b>	
2:00pm	<b>ESPN2</b> Wire to Wire
<b>SATURDAY, JUNE 15</b>	
5:00pm	<b>CBS-WKRC</b> Horse Racing Stephen Foster Handicap (L)
<b>WEDNESDAY, JUNE 19</b>	
2:00pm	<b>ESPN2</b> Wire to Wire
<b>WEDNESDAY, JUNE 26</b>	
2:00pm	<b>ESPN2</b> Wire to Wire
<b>SATURDAY, JUNE 29</b>	
3:00pm	<b>ESPN2</b> Budweiser Grand Prix of Devon
5:00pm	<b>CBS-WKRC</b> Horse Racing The Mothergoose (L) (CC)
11:00pm	<b>ESPN2</b> 2Day at the Races (L)
<b>MARITAL ARTS</b>	
<b>SATURDAY, JUNE 1</b>	
10:00pm	<b>iN2</b> World Championship Kickboxing Bad to the Bone (L)
<b>MONDAY, JUNE 3</b>	
9:00pm	<b>iN2</b> World Championship Kickboxing Bad to the Bone (R)
<b>SUNDAY, JUNE 16</b>	
9:00pm	<b>iN1</b> Ultimate Fighting Championship: Ultimate Royce Gracie
<b>MONDAY, JUNE 17</b>	
1:00am	<b>iN2</b> Ultimate Fighting Championship: Ultimate Royce Gracie
11:30pm	<b>iN2</b> Ultimate Fighting Championship: Ultimate Royce Gracie



**FIGURA 6.P9** Ejemplo del listado de acceso público

PUBLIC ACCESS	MONDAY	11:30 p.m.– Fire Ball Minstry Church of God
<b>Public Access listings for Channel 24 in all Time Warner franchises in Greater Cincinnati:</b>	6 a.m.– Sonshine Gospel Hour	12:30 a.m.– Second Peter Pente-
<b>Media Bridges Cincinnati, 1100 Race St., Cincinnati 45210. 651-4171.</b>	7 a.m.– Latter Rain Ministry	costal Church
<b>Waycross Community Media (Forest Park-Greenhills-Springfield Twp.), 2086 Waycross Road, Forest Park 45240. 825-2429.</b>	8 a.m.– Dunamis of Faith	1:30 a.m.– Road to Glory Land
<b>Intercommunity Cable Regulatory Commission, 2492 Commodity Circle, Cincinnati 45241. 772-4272.</b>	8:30 a.m.– In Jesus' Name	3:30 a.m.– Shadows of the Cross
<b>Norwood Community Television, 2020 Sherman Ave., Norwood 45212. 396-5573.</b>	9 a.m.– Happy Gospel Time TV	<b>WEDNESDAY</b>
<b>SUNDAY</b>	10 a.m.– Greek Christian Hour	6 a.m.– Pure Gospal
7 a.m.– Heart of Compassion	10:30 a.m.–Armor of God	8 a.m.– ICRC Programming
7:30 a.m.– Community Pentecostal	11 a.m.– Delhi Christian Center	8:30 a.m.– Way of the Cross
8 a.m.– ICRC Programming	Noon – Humanist Perspective	9 a.m.– Church of Christ Hour
8:30 a.m.– ICRC Programming	12:30 p.m.– Waterway Hour	10 a.m.– A Challenge of Faith
9 a.m.– St. John Church of Christ	1:30 p.m.– Country Gospel Jubilee	10:30 a.m.– Miracles Still Happen
10 a.m.– Beulah Missionary Baptist	2:30 p.m.– Know Your Government	11 a.m.– Deerfield Digest
	4:30 p.m.– House of Yisrael	11:30 a.m.– Bob Schuler
	5:30 p.m.– Living Vine Presents	Noon – Friendship Baptist Church
	6:30 p.m.– Family Dialogue	2 p.m.– Businese Talk
	7 p.m.– Goodwill Talks	2:30 p.m.– ICRC Programming
	8 p.m.– Pastor Nadie Johnson	3 p.m.– ICRC Programming
	9 p.m.– Delta Kings Barbershop Show	3:30 p.m.– Temple Fitness
	Midnight – Basement Flava 2	4 p.m.– Church of God
	1 a.m.– Total Chaos Hour	5 p.m.– Around Cincinnati
	2 a.m.– Commissioned by Christ	5:30 p.m.– Countering the Silence
	3 a.m.– From the Heart	6 p.m.– Community Report
	3:30 a.m.– Words of Farrakhan	6:30 p.m.– ICRC Programming
	4:30 a.m.– Skyward Bound	7 p.m.– Inside Springdale
		8 p.m.– ICRC Sports

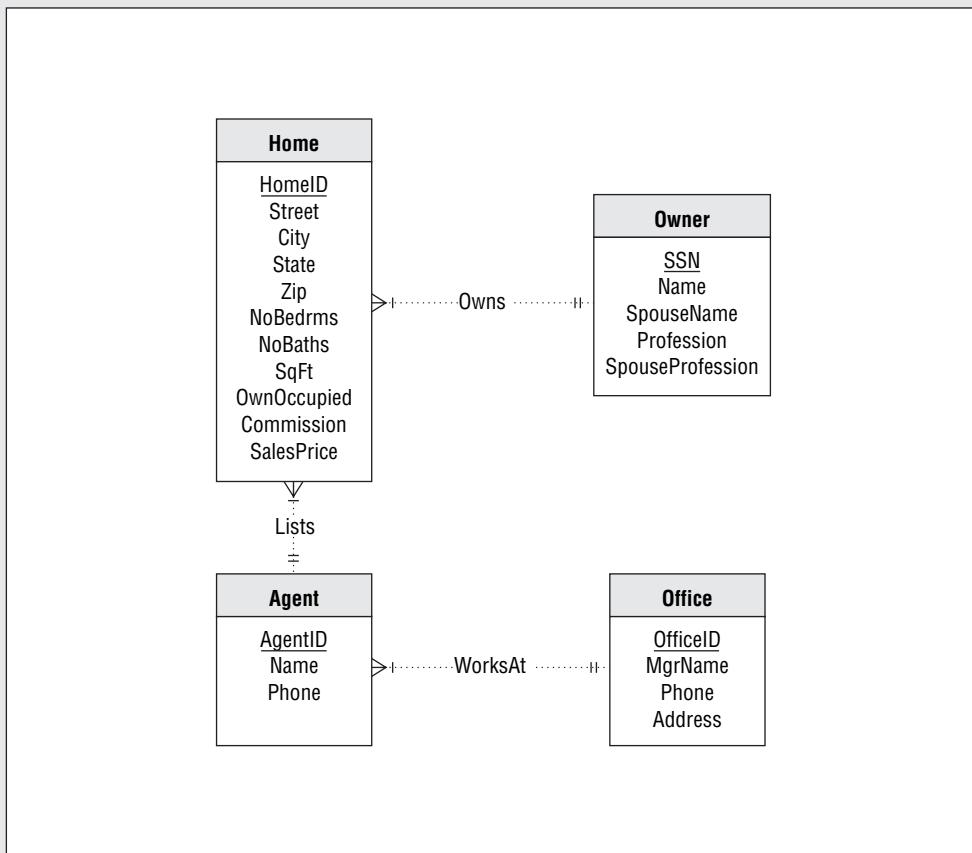
**FIGURA 6.P10** Ejemplo de la tabla de conversión

CABLE CONVERSION CHART						
Time Warner standard	upgrade cable ready	Time Warner	Insight	Fairfield, Amelia	Time Warner, Hamilton, Middletown	Adelphia Delhi
5	5	6	7	5	5	6
9	9	7	8	9	9	10
12	12	13	3	12	12	13
19	3	3	4	3	13	2
25	20	20–	25	25	20	15
48	13	8	13	6	6	8
64	11	11	11	11	11	11
2	–	–	–	–	2	–
7		–	–	–	7	–
14	14	14	14	14	14	14
16	16	16	–	16	16	16
22	–	–	–	–	8	–
43	–	–	–	–	3	–
45	–	–	–	–	10	–
54	21	21	2	–	–	4
A&E	39	39	52	28	27	46
AMC	46	46	31	29	26	40

## Problemas de conversión

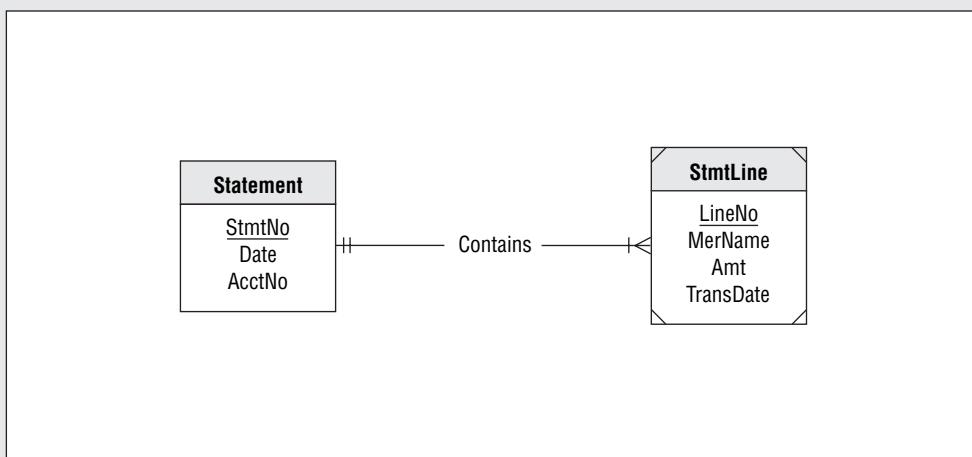
- Convierta el ERD que se muestra en la figura 6.PC1 en tablas. Enliste las reglas de conversión usadas y los cambios resultantes para las tablas.

**FIGURA 6.PC1**  
ERD para conversión  
del problema 1



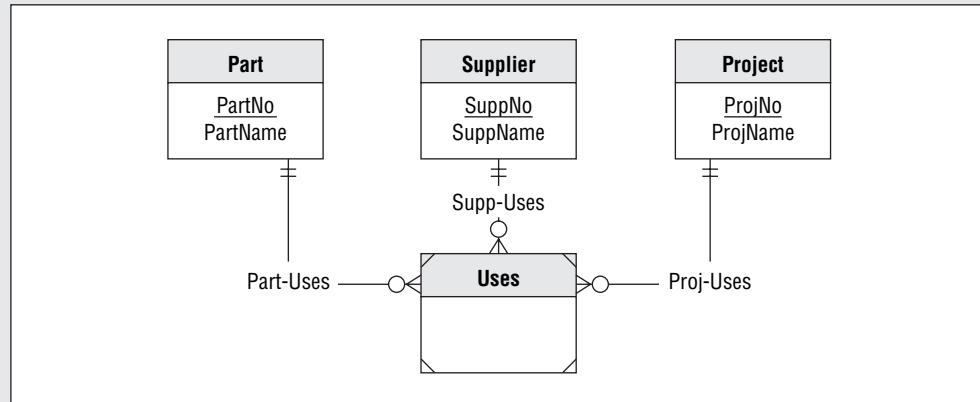
- Convierta el ERD que se muestra en la figura 6.PC2 en tablas. Enliste las reglas de conversión usadas y los cambios resultantes para las tablas.

**FIGURA 6.PC2**  
ERD para conversión  
del problema 2



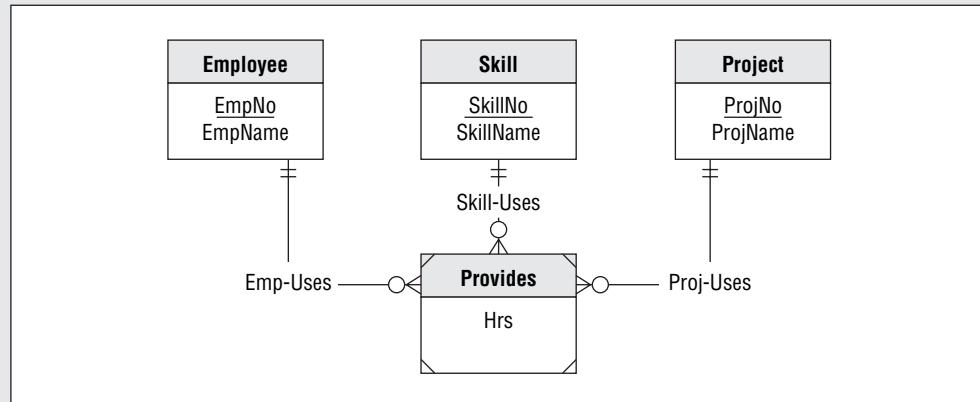
3. Convierta el ERD que se muestra en la figura 6.PC3 en tablas. Enliste las reglas de conversión usadas y los cambios resultantes para las tablas.

**FIGURA 6.PC3**  
ERD para conversión  
del problema 3



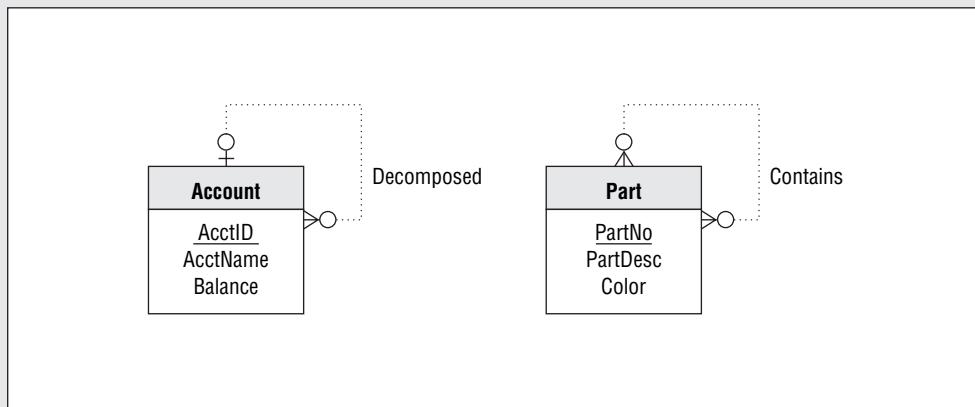
4. Convierta el ERD que se muestra en la figura 6.PC4 en tablas. Enliste las reglas de conversión usadas y los cambios resultantes para las tablas.

**FIGURA 6.PC4**  
ERD para conversión  
del problema 4



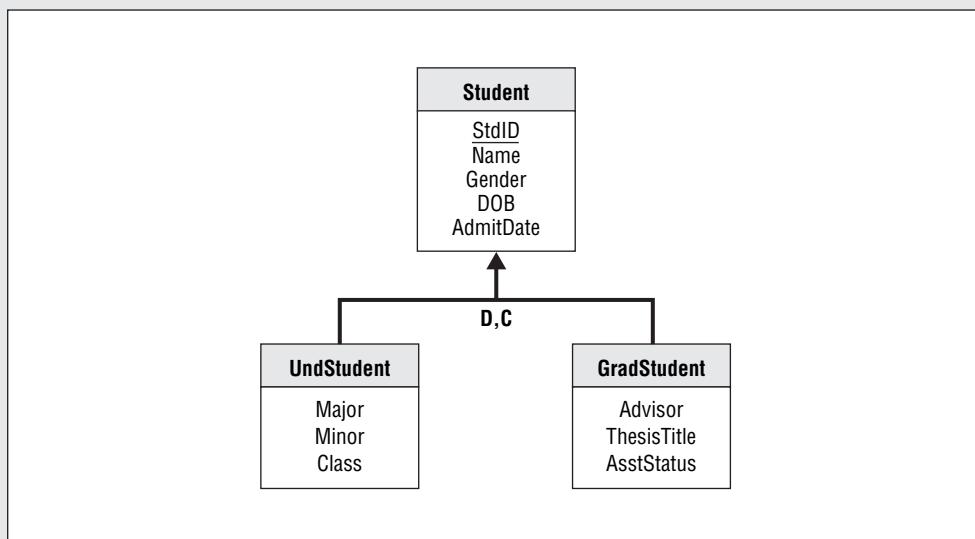
5. Convierta el ERD que se muestra en la figura 6.PC5 en tablas. Enliste las reglas de conversión usadas y los cambios resultantes a las tablas.

**FIGURA 6.PC5**  
ERD para conversión  
del problema 5



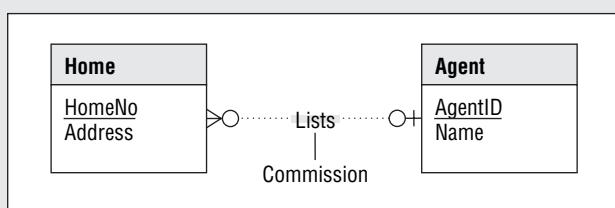
6. Convierta el ERD que se muestra en la figura 6.PC6 en tablas. Enliste las reglas de conversión usadas y los cambios resultantes para las tablas.

**FIGURA 6.PC6**  
ERD para conversión  
del problema 6



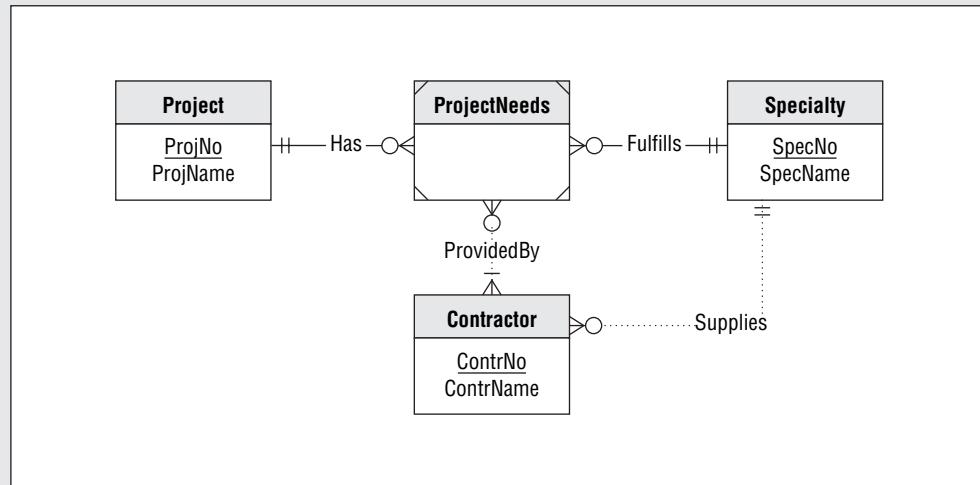
7. Convierta el ERD que se muestra en la figura 6.PC7 en tablas. Enliste las reglas de conversión usadas y los cambios resultantes para las tablas.

**FIGURA 6.PC7**  
ERD para conversión  
del problema 7



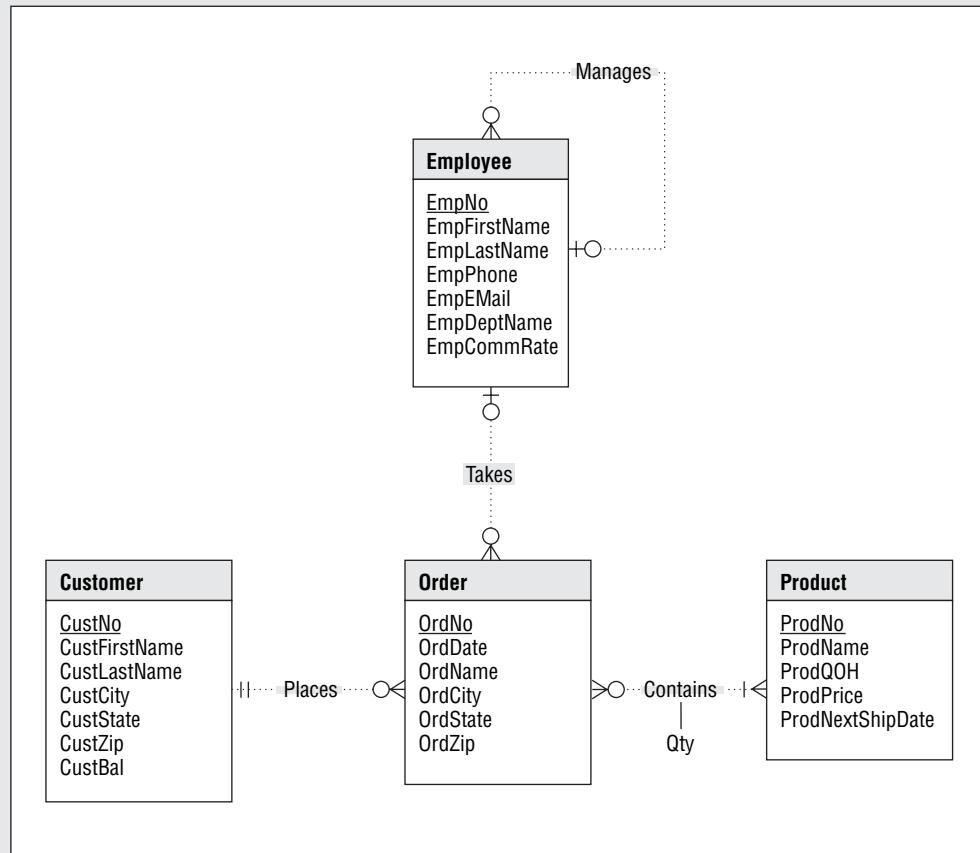
8. Convierta el ERD que se muestra en la figura 6.PC8 en tablas. Enliste las reglas de conversión usadas y los cambios resultantes a las tablas.

**FIGURA 6.PC8**  
ERD para conversión  
del problema 8



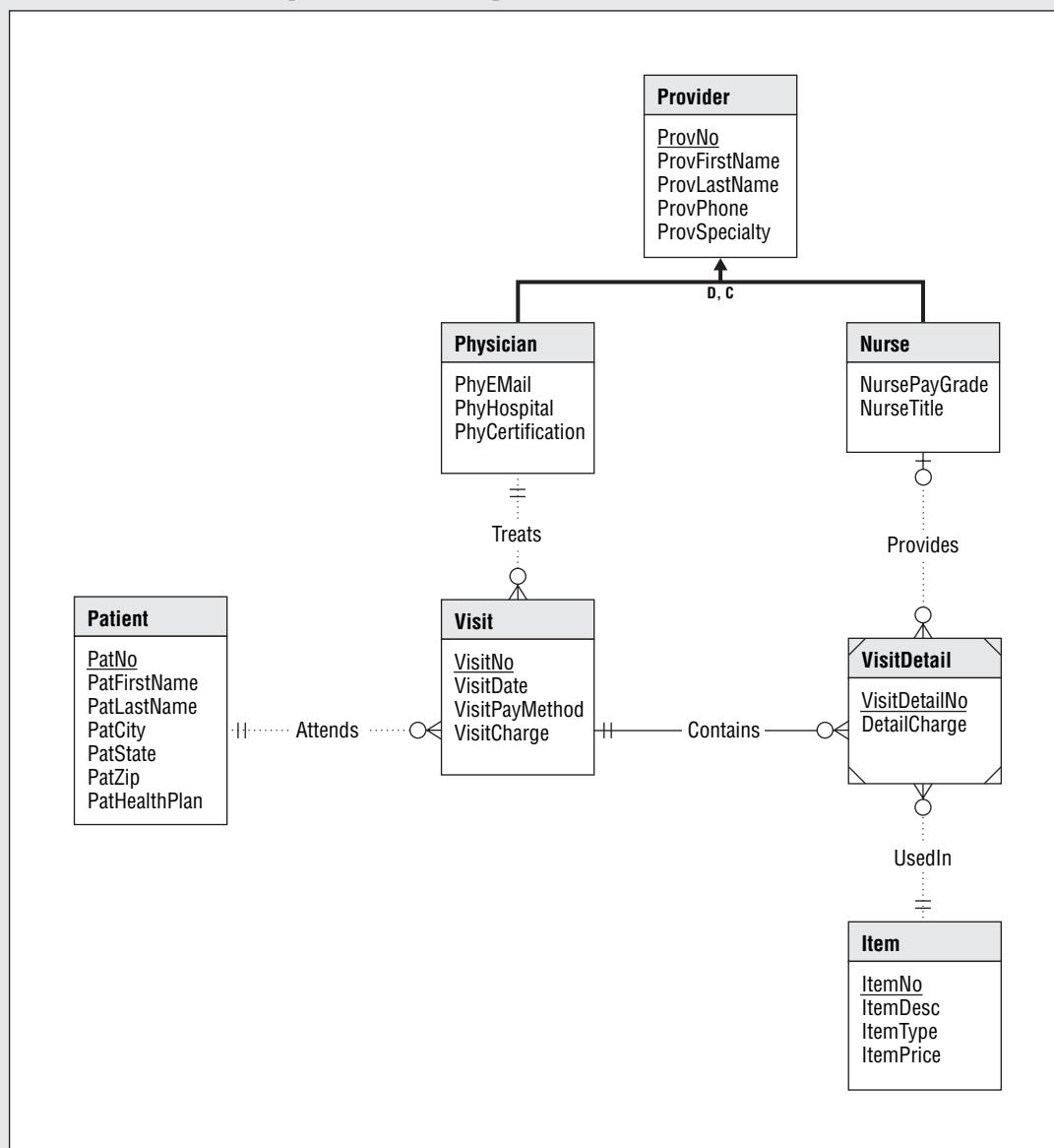
9. Convierta el ERD que se muestra en la figura 6.PC9 en tablas. Enliste las reglas de conversión usadas y los cambios resultantes para las tablas.

**FIGURA 6.PC9**  
ERD para conversión  
del problema 9



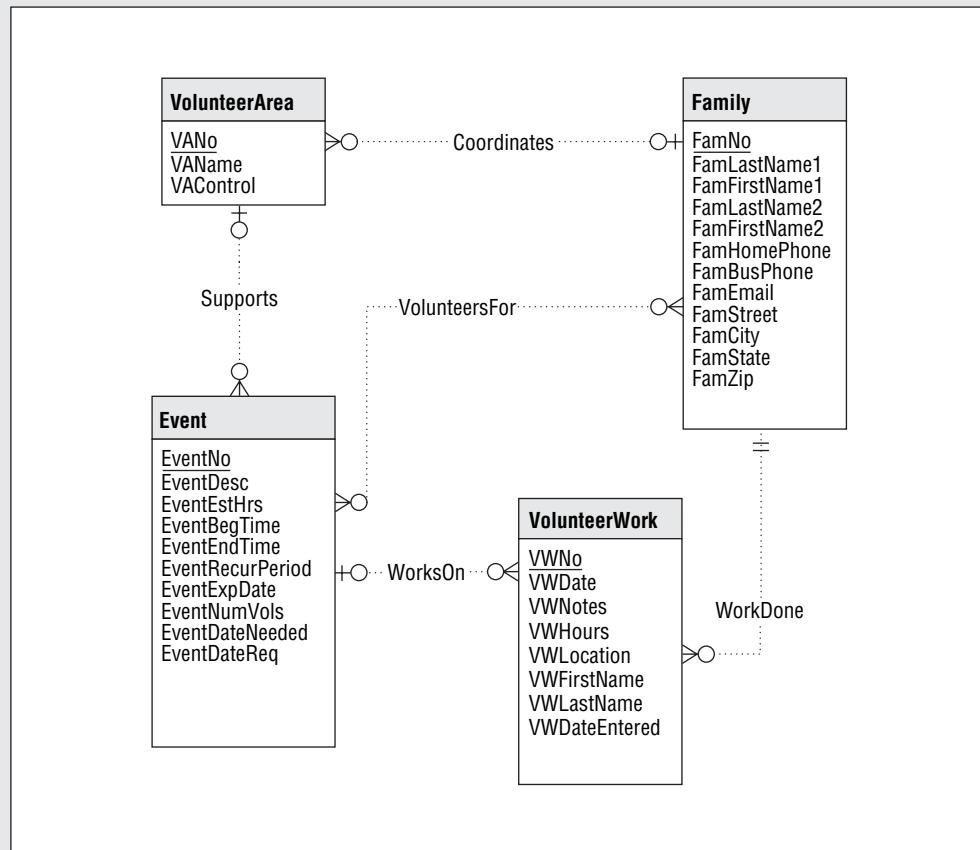
10. Convierta el ERD que se muestra en la figura 6.PC10 en tablas. Enliste las reglas de conversión usadas y los cambios resultantes para las tablas.

**FIGURA 6.PC10** ERD para conversión del problema 10



11. Convierta el ERD que se muestra en la figura 6.PC11 en tablas. Enliste las reglas de conversión usadas y los cambios resultantes para las tablas.

**FIGURA 6.PC11**  
ERD para conversión  
del problema 11



## Referencias para ampliar su estudio

El capítulo 3 de Batini, Ceri y Navathe (1992), y el capítulo 10 de Nijssen y Halpin (1989) proporcionan más detalles acerca de la transformación para refinar un ERD. Para conocer más detalles acerca de la conversión de las jerarquías de organización, consulte el capítulo 11 de Batini, Ceri y Navathe. El sitio DBAZine ([www.dbazine.com](http://www.dbazine.com)) y el sitio de DevX Database Zone ([www.devx.com](http://www.devx.com)) ofrecen consejos prácticos acerca del desarrollo de las bases de datos y del modelado de las mismas.



Parte

4

# Diseño de bases de datos relacionales

---

Los capítulos de la parte 4 ponen en práctica las habilidades y el diseño de procesos para bases de datos relacionales para permitirle implementar un diseño conceptual utilizando un DBMS relacional. El capítulo 7 abarca la motivación para la normalización de datos y proporciona una cobertura detallada de las dependencias funcionales, formularios normales y consideraciones prácticas que se aplican en la normalización de datos.

El capítulo 8 contiene una amplia cobertura del diseño físico de bases de datos, que incluye los objetivos, entradas y estructura de archivos y la optimización de consultas, junto con las guías requeridas para optar por importantes alternativas del diseño.

---

**Capítulo 7.** Normalización de tablas relacionales

**Capítulo 8.** Diseño físico de bases de datos



# Capítulo

# 7

# Normalización de tablas relacionales

## Objetivos de aprendizaje

Este capítulo describe la normalización, una técnica para eliminar la redundancia indeseada de las tablas. Al finalizar este capítulo, el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Identificar las modificaciones anormales en tablas con redundancias excesivas.
- Definir las dependencias funcionales entre las columnas de una tabla.
- Normalizar las tablas mediante la detección de violaciones a la forma normal y mediante la aplicación de las reglas de normalización.
- Analizar las relaciones M-way utilizando el concepto de independencia.
- Apreciar la utilidad y limitaciones de la normalización.

## Panorama general

Los capítulos 5 y 6 proporcionaron las herramientas para el modelado de datos, una habilidad fundamental para el desarrollo de bases de datos. Usted aprendió acerca de la notación utilizada en los diagramas de entidad-relación, patrones importantes del modelado de datos, guías para evitar los errores más comunes del modelado y conversión de los diagramas de entidad-relación (ERD) en tablas relacionales. Aplicó estos conocimientos para construir ERD simples para problemas narrativos. Este capítulo ampliará sus habilidades para el diseño de bases de datos presentando las técnicas de normalización para eliminar la redundancia en las tablas relacionales.

Las redundancias pueden ocasionar que las operaciones de inserción, actualización y eliminación generen resultados colaterales inesperados, conocidos como anomalías de modificación. Este capítulo recomienda las técnicas de normalización para eliminar las anomalías de modificación ocasionadas por las redundancias. Además, aprenderá a analizar las relaciones M-way para las redundancias. Este capítulo finaliza presentando brevemente formas normales adicionales y describiendo la utilidad y las limitaciones de las técnicas de normalización en el proceso de desarrollo de las bases de datos.

## 7.1 Panorama general del diseño de bases de datos relacionales

---

Su trabajo no termina después de convertir un ERD en tablas relacionales. Necesita analizar las tablas para encontrar las redundancias que pueden hacer que las tablas sean difíciles de utilizar. Esta sección describe por qué las redundancias pueden hacer que una tabla sea difícil de usar y presenta un tipo de restricción importante para analizar las redundancias.

### 7.1.1 Evite anomalías de modificación

**anomalías de modificación**  
un efecto colateral inesperado que ocurre cuando se modifican datos de una tabla que tiene demasiadas redundancias.

Un buen diseño de base de datos se cerciora de que los usuarios puedan modificar los contenidos de la base sin tener efectos colaterales inesperados. Por ejemplo, con una base de datos de alguna universidad, un usuario debe ser capaz de insertar un curso nuevo sin tener que agregar simultáneamente una nueva oferta del curso y un nuevo estudiante inscrito en el curso. De la misma forma, cuando se elimina un estudiante de la base de datos por haberse graduado, los datos del curso no deben perderse de forma inesperada. Estos problemas son ejemplos de anomalías de modificación, efectos colaterales que pueden ocurrir cuando se modifican los contenidos de una tabla que tiene un exceso de redundancias. Un buen diseño de base de datos evita las anomalías de modificación eliminando las redundancias excesivas.

Para comprender de forma más precisa el impacto de las anomalías de modificación, consideremos una base de datos con un pobre diseño. Imagine que la base de datos de una universidad está formada por la única tabla que se muestra la tabla 7.1. El pobre diseño de la base de datos facilita la identificación de anomalías. La siguiente lista describe algunos de los problemas del diseño.<sup>1</sup>

- Esta tabla tiene anomalías de inserción. Una anomalía de inserción se presenta cuando se agregan datos adicionales además de los requeridos por la base de datos. Por ejemplo, para insertar un curso, es necesario conocer a un estudiante y uno de los cursos que se ofrecen, ya que la llave primaria es la combinación de *StdSSN* y *OfferNo*.
- Esta tabla tiene anomalías de actualización. Una anomalía de actualización ocurre cuando es necesario modificar varias filas para modificar un solo hecho. Por ejemplo, si modificamos el atributo *StdClass* del estudiante S1, se deben modificar dos filas. Si S1 se inscribió en 10 clases, se deben modificar 10 filas.
- Esta tabla tiene anomalías de eliminación. Una anomalía de eliminación ocurre cuando se borra una fila y de forma inadvertida ocasiona que se borren otros datos. Por ejemplo, si borramos la inscripción de S2 en O3 (tercera fila), perdemos la información del curso ofrecido O3 y del curso C3.

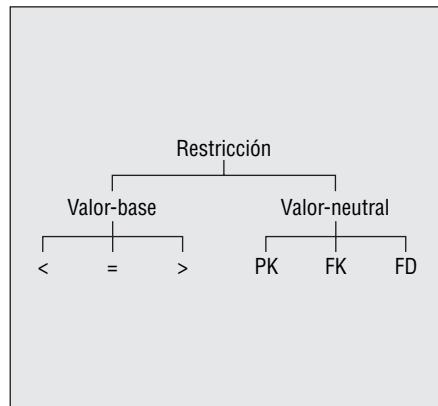
Para tratar estas anomalías, los usuarios pueden circunscribir las (por ejemplo, usar una llave primaria por *default* para insertar un curso) o los programadores de la base de datos pueden escribir código que prevenga la pérdida inadvertida de datos. Una mejor solución es modificar el diseño de la tabla para eliminar las redundancias que ocasionan las anomalías.

**TABLA 7.1 Datos de ejemplo de la tabla de la base de datos de la gran universidad**

StdSSN	StdCity	StdClass	OfferNo	OffTerm	OffYear	EnrGrade	CourseNo	CrsDesc
S1	SEATTLE	JUN	O1	FALL	2006	3.5	C1	DB
S1	SEATTLE	JUN	O2	FALL	2006	3.3	C2	VB
S2	BOTHELL	JUN	O3	SPRING	2007	3.1	C3	OO
S2	BOTHELL	JUN	O2	FALL	2006	3.4	C2	VB

<sup>1</sup> El diseño de una tabla única no es tan malo como parece. Los usuarios con un entrenamiento apropiado en bases de datos generalmente diseñan una base de datos con una sola tabla.

**FIGURA 7.1**  
Clasificación de restricciones de bases de datos



### 7.1.2 Dependencias funcionales

Las dependencias funcionales son poderosas herramientas cuando se analiza una tabla en la búsqueda de redundancias innecesarias. Una dependencia funcional es una restricción acerca de los contenidos de la base de datos. Las restricciones pueden clasificarse en valor-base *versus* valor-neutral (figura 7.1). Una restricción valor-base involucra una comparación de una columna con una constante usando un operador de comparación como  $<$ ,  $=$ , o  $>$ . Por ejemplo, edad  $\geq 21$  es una importante restricción valor-base de una base de datos utilizada para restringir la venta de alcohol a menores de edad. Una restricción valor-neutral involucra una comparación de columnas. Por ejemplo, una restricción valor-neutral sería que la edad de jubilación debe ser mayor que la edad actual en una base de datos de planeación de jubilaciones.

Las restricciones de llave primaria (PK) y llave foránea (FK) son importantes tipos de restricciones valor-neutral. Una llave primaria puede tener cualquier valor mientras no coincida con alguna llave primaria de una fila existente. Una restricción de llave foránea requiere que el valor de una columna de una tabla coincida con el valor de la llave primaria en otra tabla.

Una dependencia funcional es otro tipo importante de restricción de valor-neutral. Una dependencia funcional (FD, por sus siglas en inglés) es una restricción de dos o más columnas de una tabla.  $X$  determina  $Y$  ( $X \rightarrow Y$ ) si existe al menos un valor de  $Y$  para cada valor de  $X$ . La palabra función proviene de las matemáticas en donde una función proporciona un valor. Por ejemplo, el número de seguridad social determina la ciudad ( $StdSSN \rightarrow StdCity$ ) en la tabla de la base de datos de la universidad si hay al menos un valor de ciudad para número de seguridad social. A una columna que aparece del lado izquierdo de la FD se le llama determinante u, opcionalmente, LHS (left-hand side) por las siglas en inglés correspondientes a mano-izquierda. En este ejemplo,  $StdSSN$  es un determinante.

También puede pensar en las dependencias funcionales como identificación de llaves candidatas potenciales. Al establecer que  $X \rightarrow Y$ , si  $X$  y  $Y$  se colocan juntas en una tabla sin otras columnas,  $X$  es una llave candidata. Cada determinante (LHS) es una llave candidata en el caso de que se le coloque en una tabla con otras columnas que determine. Por ejemplo, si  $StdSSN$ ,  $StdCity$  y  $StdClass$  se colocan juntas en una tabla y  $StdSSN \rightarrow StdCity$  y  $StdSSN \rightarrow StdClass$ , entonces,  $StdSSN$  es una llave candidata. Si no existen otras llaves candidatas, una determinante se convertirá en la llave primaria siempre y cuando no permita valores nulos.

#### Listas y diagramas de dependencias funcionales

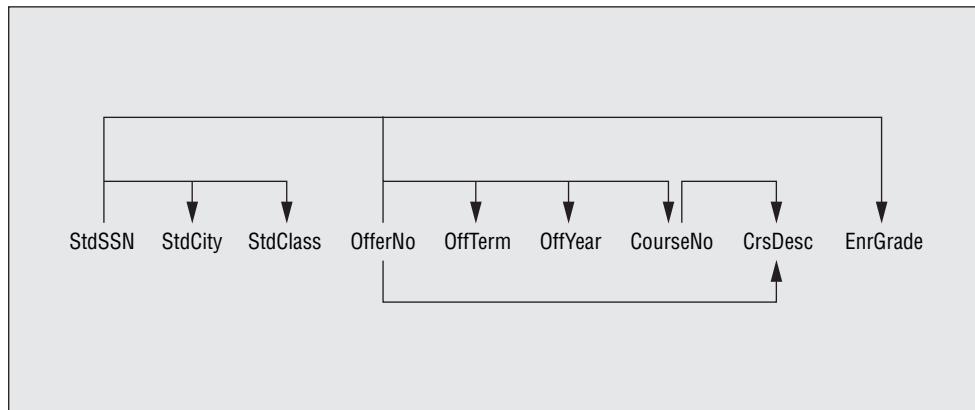
Un diagrama de dependencia funcional despliega de forma muy compacta las dependencias funcionales de una tabla en particular. Debe acomodar las FD para agrupar de forma visual las columnas que comparten el mismo determinante. En la figura 7.2 es fácil identificar las dependencias en las que  $StdSSN$  es el determinante. Al examinar la posición y la altura de las líneas, puede observar que la combinación de  $StdSSN$  y  $OfferNo$  determina  $EnrGrade$ , mientras que  $OfferNo$  determina, por sí mismo,  $OfferTerm$ ,  $OfferYear$  y  $CourseNo$ . Un buen diagrama visual puede facilitar el proceso de normalización que se describe en la siguiente sección.

#### dependencia funcional

una restricción que se aplica a dos o más columnas de una tabla.  $X$  determina  $Y$  ( $X \rightarrow Y$ ) si existe al menos un valor de  $Y$  para cada valor de  $X$ .

**FIGURA 7.2**

**Diagrama de dependencias para la gran tabla de base de datos de la universidad**

**TABLA 7.2**

**Lista de FD para la tabla de base de datos de la universidad**

StdSSN→→StdCity, StdClass
OfferNo→→OffTerm, OffYear, CourseNo, CrsDesc
CourseNo→→CrsDesc
StdSSN, OfferNo→→EnrGrade

Si lo prefiere puede enlistar las FD en lugar de acomodarlas en un diagrama. Para grandes colecciones de FD, es difícil hacer un diagrama. Debe listar los FD, agrupados por LHS, tal como se muestra en la tabla 7.2.

#### *Identificación de dependencias funcionales*

Además de comprender la definición de la dependencia funcional y la notación, los diseñadores de bases de datos deben ser capaces de identificar las dependencias funcionales cuando reúnen los requerimientos de bases de datos. En el planteamiento textual de los problemas se pueden identificar algunas dependencias funcionales al plantear enunciados que indiquen unicidad. Por ejemplo, un usuario pudiera indicar que cada curso ofertado tiene un único número de oferta exclusivo, junto con el año y el fin de la oferta. A partir de este enunciado, el diseñador debe inducir que  $OfferNo \rightarrow OffYear$  y  $OfferNo \rightarrow OffTerm$ . También puede encontrar las dependencias funcionales de un diseño de tablas, resultado de la conversión de un ERD. Las dependencias funcionales serán inducidas por cada columna única (llave primaria u otra llave candidata) con la única columna como LHS y el resto de las columnas de la tabla en el RHS (*right-hand side*; a mano derecha).

Aunque las dependencias funcionales derivadas de los enunciados de exclusividad sean fáciles de identificar, las dependencias funcionales derivadas de los enunciados de las relaciones 1-M pueden ser confusas al momento de identificarlos. Cuando observe un enunciado de relación 1-M, la dependencia funcional se deriva en la dirección hija-madre, y no en la dirección madre-hija. Por ejemplo, el enunciado “Un catedrático imparte muchos cursos, pero un curso es impartido por un sólo catedrático” define una dependencia funcional desde una columna única para los cursos hasta una sola columna de catedráticos, tal como  $OfferNo \rightarrow FacNo$ . Los diseñadores novatos algunas veces imponen de forma incorrecta que  $FacNo$  determina una colección de valores de  $OfferNo$ . Este enunciado no es correcto, ya que una dependencia funcional debe permitir como máximo un sólo valor asociado, no una colección de valores.

Las dependencias funcionales en las que el LHS no es una llave primaria o candidata también pueden ser difíciles de encontrar. Estas FD son especialmente importantes de identificar después de convertir un ERD en un diseño de tablas. Debe buscar cuidadosamente las FD en las que el LHS no sea una llave primaria o candidata. También debe considerar las FD de las tablas, con una llave primaria o candidata combinada, en donde el LHS sea parte de una llave pero no de la llave completa. La presentación de los formularios normales de la sección 7.2 explica que estos tipos de FD pueden llevar a anomalías de modificación.

#### **FD para las relaciones 1-M**

imponer una FD en la dirección hija-madre de una relación 1-M. No imponga una FD para la dirección madre-hija, ya que cada valor LHS se puede asociar como máximo con un valor RHS.

**determinante mínimo**  
el determinante [columna(s) que aparece a mano izquierda de una dependencia funcional] no debe incluir columnas adicionales. Este requerimiento de minimalismo es semejante al requerimiento de minimalismo de las llaves candidatas.

**eliminación de FD potenciales**  
utilizando muestras de datos para eliminar FD potenciales. Si dos columnas tienen el mismo valor para el LHS pero distintos valores para el RHS, no puede existir una FD. Algunos programas comerciales de normalización usan esta técnica para ayudarle al usuario a determinar las FD.

Otra consideración importante en la imposición de dependencias funcionales es el minimalismo del LHS. Es importante distinguir cuando una columna por sí sola es la determinante *versus* una combinación de columnas. Una FD en la que el LHS incluya más de una columna generalmente representa una relación M-N. Por ejemplo, el enunciado “El monto de la orden se cobra para cada producto comprado en una orden”, se traduce en la FD  $OrdNo, ProdNo \rightarrow OrdQty$ . El monto de la orden depende de la combinación del número de orden y del número de producto, no sólo de una de las dos columnas.

Parte de la confusión acerca del minimalismo del LHS se debe al significado de las columnas que están a mano izquierda *versus* las que están a mano derecha de una dependencia. Para registrar que el número de seguridad social de un estudiante determina la ciudad y la clase, usted puede escribir tanto  $StdSSN \rightarrow StdCity, StdClass$  (de forma más compacta) o  $StdSSN \rightarrow StdCity$  y  $StdSSN \rightarrow StdClass$  (de forma más extensa). Si considera que la dirección de correo electrónico también es única para cada estudiante, puede escribir  $Email \rightarrow StdCity, StdClass$ . No deberá escribir  $StdSSN, Email \rightarrow StdCity, StdClass$ , ya que estas FD implican que la combinación de  $StdSSN$  y  $Email$  son el determinante. Por ende, debe escribir las FD para que LHS no tenga columnas que no sean necesarias.<sup>2</sup> La prohibición en contra de las columnas innecesarias para los determinantes es la misma prohibición en contra de las columnas innecesarias para las llaves candidatas. Tanto los determinantes como las llaves candidatas deben ser mínimos.

#### *Eliminación de FD mediante una muestra de datos*

No se puede probar la existencia de una dependencia funcional al examinar las filas de una tabla. Sin embargo, puede falsificar una dependencia funcional al examinar los contenidos de una tabla (pruebe, por ejemplo, que una dependencia funcional en realidad no existe). Por ejemplo, en la base de datos de la universidad (tabla 7.1) concluimos que  $StdClass$  no determina  $StdCity$ , ya que hay dos filas con el mismo valor para  $StdClass$  pero un valor distinto para  $StdCity$ . Por tanto, algunas veces resulta útil examinar muestras de las filas en una tabla para eliminar las potenciales dependencias funcionales. Existen diversas herramientas comerciales de diseño de bases de datos que automatizan el proceso de eliminar dependencias a través del análisis de filas muestreadas. Finalmente, es el diseñador de bases de datos quien debe tomar la decisión final acerca de las dependencias funcionales que existan en una tabla.

## 7.2 Formas normales

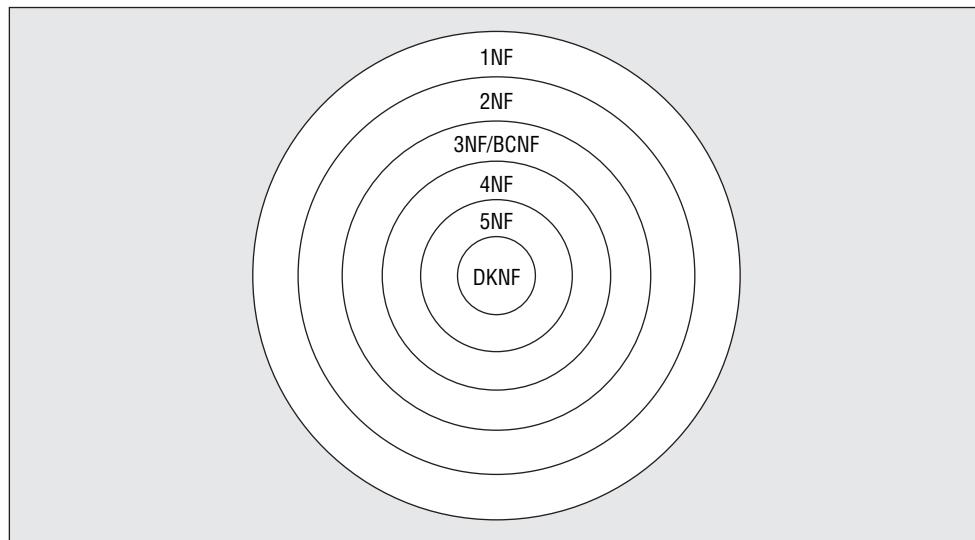
---

La normalización es el proceso de eliminación de redundancias en una tabla para que sea más fácil de modificar. Se han desarrollado un sinnúmero de formas normales para eliminar las redundancias. Una forma normal es una regla sobre las dependencias permisibles. Cada forma normal elimina cierto tipo de redundancias. Tal como se muestra en la figura 7.3, el punto de partida es la primera forma normal (1NF). Todas las tablas sin grupos repetidos se encuentran en la 1NF. 2NF es más rígida que 1NF. Sólo un subconjunto de las tablas de 1NF se encuentra en 2NF. Cada forma normal sucesiva refina la forma normal previa para eliminar otro tipo de redundancia adicional. Debido a que la BCNF (forma normal de Boyce-Codd, por sus siglas en inglés) es una definición revisada (y más rígida) para 3NF, la 3NF y la BCNF se muestran en la misma parte de la figura 7.3.

2NF y 3NF/BCNF son reglas acerca de las dependencias funcionales. Si las dependencias funcionales de una tabla coinciden con un patrón específico, la tabla se encuentra en la forma normal especificada. 3NF/BCNF es la parte más importante en la práctica porque las formas normales superiores involucran otros tipos de dependencias que son menos comunes y más difíciles de entender. Por lo mismo, se pone mayor énfasis en 3NF/BCNF. La sección 7.3 presenta 4NF como una manera de razonar acerca de las relaciones M-way. La sección 7.4 presenta 5NF y DKNF (forma normal del dominio de la llave, por sus siglas en inglés) para mostrar que se han propuesto formas normales superiores. DKNF es la máxima forma normal, pero queda más en un ideal que en una forma normal práctica.

<sup>2</sup> A este concepto se le conoce de forma más apropiada como “dependencia funcional total”. La dependencia funcional total significa que el LHS es mínimo.

**FIGURA 7.3**  
Relación de los formularios normales



**TABLA 7.3 Tabla desnormalizada de la base de datos de la universidad**

StdSSN	StdCity	StdClass	OfferNo	OffTerm	OffYear	EnrGrade	CourseNo	CrsDesc
S1	SEATTLE	JUN	O1	FALL	2006	3.5	C1	DB
			O2	FALL	2006	3.3	C2	VB
S2	BOTHELL	JUN	O3	SPRING	2007	3.1	C3	OO
			O2	FALL	2006	3.4	C2	VB

### 7.2.1 Primera forma normal

La 1NF prohíbe la anidación o repetición de grupos en las tablas. Una tabla que no esté en 1NF está desnormalizada o sin normalizar. En la tabla 7.3, la tabla de la universidad está desnormalizada porque los dos renglones contienen grupos repetidos o tablas anidadas. Para convertir una tabla desnormalizada en 1NF, reemplace cada valor de un grupo repetido por una fila. En una fila nueva, usted copia las columnas que no se repiten. Puede observar la conversión al comparar la tabla 7.3 con la tabla 7.1 (dos filas con grupos repetidos *versus* cuatro filas sin grupos repetidos).

Debido a que la mayoría de los DBMS comerciales requieren tablas en 1NF, regularmente no necesitará convertir las tablas a 1NF.<sup>3</sup> Sin embargo, comúnmente necesitará llevar a cabo el proceso inverso (tablas en 1NF a tablas desnormalizadas) para generar reportes. Tal como se describe en el capítulo 10, los reportes usan la anidación para mostrar las relaciones. Sin embargo, las tablas referidas no tienen anidación.

### 7.2.2 Segunda y tercera forma normal

Las definiciones de 2NF y 3NF hacen la distinción entre columnas llave y columnas no llave.<sup>4</sup> Una columna es una columna llave si es parte de una llave candidata o una llave candidata en sí misma. Recuerde que una llave candidata es un conjunto mínimo de columna(s) con valores únicos en la tabla. Mínimo significa que ninguna de las columnas se puede eliminar sin eliminar la propiedad de exclusividad. Las columnas que no son llave son cualquier otra columna. En

<sup>3</sup> Aunque las tablas anidadas se incluyen desde el estándar SQL:1999 con soporte comercial en Oracle 9i, esta característica parece no ser importante en la mayoría de las aplicaciones de negocios. Por ende, este capítulo no considera las complicaciones de las tablas anidadas en la normalización.

<sup>4</sup> En la literatura, a las columnas llave se les conoce como principales y a las columnas no llave se les conoce como no principales.

**definición combinada de 2NF y 3NF**

una tabla está en 3NF si cada columna que no forma parte de la llave depende de todas las llaves candidatas, llaves candidatas completas, y nada más que llaves candidatas.<sup>5</sup>

**definición 2NF**

una tabla está en 2NF si cada columna que no forma parte de la llave depende de todas las llaves candidatas, no de un subconjunto de cualquier llave candidata.

**violación 2NF**

una FD en la que parte de la llave determina a una no llave viola la 2NF. Una FD con una sola columna LHS no puede violar la 2NF.

la tabla 7.1, la combinación de (*StdSSN*, *OfferNo*) es la única llave candidata. Otras columnas como *StdCity* y *StdClass* son columnas que no forman parte de la llave.

El objetivo de 2NF y 3NF es generar tablas en las que cada llave determine el resto de las columnas. Al margen se muestra una forma fácil de recordar las definiciones 2NF y 3NF.

*Segunda forma normal*

Para comprender esta definición, separémosla en las partes 2NF y 3NF. La definición 2NF usa la primera parte de la definición tal como se muestra en el margen.

Para revisar si una tabla está en 2NF, debe buscar las FD que violen la definición. Una FD en la que parte de la llave determine una columna no llave viola la 2NF. Si la llave contiene una sola columna, la tabla está en 2NF. Observando el diagrama de dependencias de la figura 7.2, puede detectar fácilmente las violaciones hacia la 2NF. Por ejemplo, *StdCity* es una columna que no forma parte de la llave, pero es *StdSSN* y no la llave primaria completa (combinación de *StdSSN* y *OfferNo*) la que lo determina. Las únicas FD que satisfacen la definición de 2NF son *StdSSN*, *OfferNo* → *EnrGrade* y *CourseNo* → *CrsDesc*.

Para colocar la tabla en 2NF, separe la tabla original en pequeñas tablas que satisfagan la definición 2NF. En cada una de las tablas pequeñas, la llave primaria completa (no parte de la llave primaria) debe determinar las columnas que no forman parte de la llave. El proceso de separación incluye al operador de proyección del álgebra relacional. Para la tabla de la base de datos de la universidad, tres operaciones de proyección la separan de tal forma que la llave primaria determina las columnas que no forman parte de la llave en cada una de las tablas que se muestran abajo.

**UnivTable1** (*StdSSN*, *StdCity*, *StdClass*)

**UnivTable2** (*OfferNo*, *OffTerm*, *OffYear*, *CourseNo*, *CrsDesc*)

**UnivTable3** (*StdSSN*, *OfferNo*, *EnrGrade*)

El proceso de separación debe preservar la tabla original de dos formas. Primero, la tabla original debe poderse recuperar usando operaciones de enlaces naturales sobre las tablas más pequeñas. Segundo, las FD de la tabla original deben poderse deducir a partir de las FD de las tablas más pequeñas. Técnicamente, al proceso de separación se le conoce como sin pérdida, descomposición de tipo preservación-dependencia. Algunas de las referencias al final de este capítulo explican la teoría detrás del proceso de separación.

Después de separar la tabla original en tablas más pequeñas, debe agregar las restricciones de integridad referencial para conectar las tablas. En cuanto se separe una tabla, la columna que se separa se convierte en una llave foránea en la tabla en la que no es una llave primaria. Por ejemplo, *StdSSN* es una llave foránea en *UnivTable3* dado que la tabla original de la universidad se separó en esta columna. Por lo mismo, define una restricción de integridad referencial que establece que *UnivTable3.StdSSN* hace referencia a *UnivTable1.StdSSN*. La tabla *UnivTable3* se repite más abajo con sus restricciones de integridad referencial.

**UnivTable3** (*StdSSN*, *OfferNo*, *EnrGrade*)

FOREIGN KEY (*StdSSN*) REFERENCES *UnivTable1*

FOREIGN KEY (*OfferNo*) REFERENCES *UnivTable2*

*Tercera forma normal*

*UnivTable2* todavía tiene anomalías de la modificación. Por ejemplo, usted no puede agregar un curso nuevo a menos que se conozca el valor de la columna *OfferNo*. Para eliminar las anomalías de la modificación, se debe aplicar la definición de 3NF.

Una FD en la que una columna que no forma parte de la llave determine a otra columna que no forma parte de la llave viola la 3NF. En el caso de *UnivTable2* arriba, la FD (*CourseNo* → *CrsDesc*) viola la 3NF porque ambas columnas, *CourseNo* y *CrsDesc* son columnas que no

<sup>5</sup> Puede recordar esta definición mediante su analogía con el juramento tradicional de la justicia: “¿Jura decir la verdad, toda la verdad y nada más que la verdad...?”

forman parte de la llave. Para subsanar esta violación, separe *UnivTable2* en dos tablas, tal como se muestra más abajo, y agregue una restricción de llave foránea.

**UnivTable2-1** (OfferNo, OffTerm, OffYear, CourseNo)  
 FOREIGN KEY (CourseNo) REFERENCES UnivTable2-2  
**UnivTable2-2** (CourseNo, CrsDesc)

### dependencia transitiva

una FD derivada por la ley de la transitividad. Las FD transitivas no deben registrarse como insumos del proceso de normalización.

Una forma equivalente de definir la 3NF es que 3NF prohíbe las dependencias transitivas. Una dependencia transitiva es una dependencia funcional derivada por la ley de la transitividad. La ley de la transitividad dice que si un objeto A se relaciona con B y B se relaciona con C, entonces, se puede concluir que A se relaciona con C. Por ejemplo, el operador  $<$  obedece la ley de la transitividad:  $A < B$  y  $B < C$  implica que  $A < C$ . Las dependencias funcionales, como el operador  $<$ , obedecen la ley de transitividad:  $A \rightarrow B$ ,  $B \rightarrow C$ , entonces,  $A \rightarrow C$ . En la figura 7.2,  $OfferNo \rightarrow CrsDesc$  es una dependencia transitiva derivada de  $OfferNo \rightarrow CourseNo$  y  $CourseNo \rightarrow CrsDesc$ .

Debido a que las dependencias transitivas se sobrepasan fácilmente, la definición sugerida de 3NF no usa dependencias transitivas. Además, en la sección 7.2.4 aprenderá que debe omitir de su análisis las dependencias derivadas tales como las dependencias transitivas.

### Ejemplo combinado de 2NF y 3NF

La gran tabla de pacientes, como se ilustra en la tabla 7.4, proporciona otro ejemplo en donde puede aplicar su conocimiento de 2NF y 3NF. La gran tabla de pacientes contiene hechos sobre pacientes, proveedores de servicios de salud, visitas de los pacientes a las clínicas y diagnósticos hechos por los proveedores de servicios de salud. La gran tabla de pacientes contiene una llave primaria combinada formada por la combinación de *VisitNo* y *ProvNo* (número de proveedor). Al igual que la gran tabla de la base de datos de la universidad ilustrada en la tabla 7.1, la gran tabla de pacientes refleja un diseño pobre de tabla con muchas redundancias. La tabla 7.5 lista las FD asociadas. Usted debe verificar que las filas muestradas de la tabla 7.4 no contradigan las FD.

Tal como se comentó en ocasiones anteriores, las FD que violan la 2NF involucran una parte de una llave que determina a una columna que no forma parte de la llave. Muchas de las FD de la tabla 7.5 violan la definición de 2NF, ya que la llave primaria es la combinación de *VisitNo* y *ProvNo*. Por ende, las FD que tengan sólo *VisitNo* o *ProvNo* en LHS violan la 2NF. Para subsanar las violaciones hacia 2NF, separe la gran tabla de pacientes para que las FD que la violentan se asocien con tablas separadas. La lista de tablas revisadas, *PatientTable1* y *PatientTable2* contiene las FD que violentan. *PatientTable3* conserva el resto de las columnas.

**TABLA 7.4** Datos de muestra de la gran tabla de pacientes

VisitNo	VisitDate	PatNo	PatAge	PatCity	PatZip	ProvNo	ProvSpecialty	Diagnosis
V10020	1/13/2007	P1	35	DENVER	80217	D1	INTERNIST	EAR INFECTION
V10020	1/13/2007	P1	35	DENVER	80217	D2	NURSE PRACTITIONER	INFLUENZA
V93030	1/20/2007	P3	17	ENGLEWOOD	80113	D2	NURSE PRACTITIONER	PREGNANCY
V82110	1/18/2007	P2	60	BOULDER	85932	D3	CARDIOLOGIST	MURMUR

**TABLA 7.5**

**Lista de FD para la gran tabla de pacientes**

PatNo  $\rightarrow$  PatAge, PatCity, PatZip  
 PatZip  $\rightarrow$  PatCity  
 ProvNo  $\rightarrow$  ProvSpecialty  
*VisitNo*  $\rightarrow$  PatNo, VisitDate, PatAge, PatCity, PatZip  
*VisitNo*, ProvNo  $\rightarrow$  Diagnosis

**PatientTable1** (ProvNo, ProvSpecialty)

**PatientTable2** (VisitNo, VisitDate, PatNo, PatAge, PatCity, PatZip)

**PatientTable3** (VisitNo, ProvNo, Diagnosis)

FOREIGN KEY (VisitNo) REFERENCES PatientTable2

FOREIGN KEY (ProvNo) REFERENCES PatientTable1

*PatientTable1* y *PatientTable3* están en 3NF, ya que no existen columnas que no forman parte de la llave y que determinen a otras columnas que no forman parte de la llave. Sin embargo, *PatientTable2* viola 3NF porque las FD  $\text{PatNo} \rightarrow \text{PatZip}$ ,  $\text{PatAge}$  y  $\text{PatZip} \rightarrow \text{PatCity}$  incluyen columnas que no forman parte de la llave que determinan a otras columnas que no son llave. Para solucionar las violaciones a 3NF, separe *PatientTable2* en tres tablas como se muestra en la lista de tablas revisada. En la lista de tablas revisada, *PatientTable2-1* y *PatientTable2-2* contienen los FD que ocasionan las violaciones, mientras que *PatientTable2-3* conserva el resto de las columnas.

**PatientTable2-1** (PatNo, PatAge, PatZip)

FOREIGN KEY (PatZip) REFERENCES PatientTable2-2

**PatientTable2-2** (PatZip, PatCity)

**PatientTable2-3** (VisitNo, PatNo, VisitDate)

FOREIGN KEY (PatNo) REFERENCES PatientTable2-1

El uso de 2NF y 3NF requiere de dos pasos de normalización. El proceso de normalización se puede realizar en un solo paso con el uso de la forma normal de Boyce-Codd, tal como se presenta en la siguiente subsección.

### 7.2.3 Forma normal de Boyce-Codd

**definición BCNF**  
una tabla está en BCNF si cada determinante es una llave candidata.

La definición revisada sobre 3NF, conocida como forma normal de Boyce-Codd (BCNF), es una mejor definición porque es más sencilla y abarca un caso especial que se omite en la definición original de 3NF. La definición BCNF es más simple porque no hace ninguna referencia a 2NF.

Las violaciones de BCNF incluyen FD en las que el determinante (LHS) no es una llave candidata. En un pobre diseño de tablas como el de la gran tabla de la base de datos de la universidad (los datos del ejemplo están en la tabla 7.1 y la lista de FD en la tabla 7.2), usted puede detectar fácilmente las violaciones hechas sobre BCNF. Por ejemplo, *StdSSN* es un determinante, pero no una llave candidata (es parte de una llave candidata, pero no una llave candidata por sí misma). La única FD de la tabla 7.2 que no viola BCNF es *StdSSN*, *OfferNo*  $\rightarrow$  *EnrGrade*.

Para revisar otro ejemplo, apliquemos la definición BCNF a las FD de la gran tabla de pacientes que se muestra en la tabla 7.5. Todas las FD de la tabla 7.5 violan la definición de BCNF a excepción de la última FD (*VisitNo*, *ProvNo*  $\rightarrow$  *Diagnosis*). Todas las demás FD tienen determinantes que no son llaves candidatas (en algunos casos son parte de la llave candidata pero no son la llave candidata completa). Para solucionar las violaciones contra BCNF, separe la tabla de pacientes en tablas más pequeñas. Cada determinante debe colocarse en una tabla separada junto con las columnas que determina. El resultado es idéntico a la separación de 3NF (vea el resultado del último ejemplo de 3NF) con las tablas *PatientTable1*, *PatientTable3*, *PatientTable2-1*, *PatientTable2-2* y *PatientTable2-3*.

#### Relación entre 3NF y BCNF

Aunque por lo general BCNF y 3NF generan el mismo resultado, BCNF es una definición más rígida que 3NF. Por ende, cada tabla dentro de BCNF está, por definición, en 3NF. BCNF abarca dos casos especiales que 3NF no cubre: (1) la parte de una llave determina parte de una llave, y (2) una columna que no forma parte de la llave determina parte de una llave. Estas situaciones sólo se dan cuando existen múltiples llaves candidatas compuestas (llaves candidatas con varias columnas). El análisis de las dependencias de las tablas con múltiples llaves candidatas compuestas es difícil. Afortunadamente, no son comunes las tablas con múltiples llaves candidatas compuestas.

*UnivTable4* ilustra una tabla en 3NF pero no en BCNF de acuerdo con la primera excepción (la parte de una llave determina parte de una llave). La tabla *UnivTable4* (figura 7.4) tiene dos llaves candidatas: la combinación de *StdSSN* y *OfferNo* (la llave primaria) y la combinación de *Email* y *OfferNo*. En las FD de la tabla *UnivTable4* (figura 7.4), debe identificar que *StdSSN* y *Email* se determinan entre ellas mismas. Dadas las FD entre *StdSSN* y *Email*, *UnivTable4* contiene una redundancia, ya que el *Email* se repite para cada *StdSSN*. Por ejemplo, las primeras dos columnas contienen la misma dirección de correo electrónico ya que el valor de *StdSSN* es el mismo. Los siguientes puntos explican por qué la tabla *UnivTable4* está en 3NF y no en BCNF:

- **3NF:** *UnivTable4* está en 3NF, ya que la única columna que no forma parte de la llave (*EnrGrade*) depende de cada una de las llaves candidatas (no sólo de una parte de una llave candidata). Dado que *EnrGrade* es la única columna que no forma parte de la llave, no puede depender de otras columnas que no formen parte de la llave.
- **BCNF:** Las dependencias entre *StdSSN* y *Email* violan BCNF. Tanto *StdSSN* como *Email* son determinantes, pero ninguna es una llave candidata completa aunque cada una forma parte de una llave candidata. Para eliminar la redundancia debe separar la tabla *UnivTable4* en dos tablas, como se muestra en la figura 7.4.

*UnivTable5* (figura 7.5) ilustra otro ejemplo de una tabla con múltiples llaves candidatas compuestas. Como *UnivTable4*, *UnivTable5* que está en 3NF pero no en BCNF, ya que parte de una llave determina parte de una llave. *UnivTable5* tiene dos llaves candidatas: la combinación de *StdSSN* y *AdvisorNo* (la llave primaria) y la combinación de *StdSSN* y *Major*. *UnivTable5* tiene una redundancia dado que *Major* se repite para cada fila que tenga el mismo valor en *AdvisorNo*. Los siguientes puntos explican por qué *UnivTable5* está en 3NF pero no en BCNF.

**FIGURA 7.4**  
Filas de ejemplo,  
diagrama de  
dependencias y tablas  
normalizadas de  
*UnivTable4*

<b>UnivTable4</b>			
<u>StdSSN</u>	<u>OfferNo</u>	<u>Email</u>	<u>EnrGrade</u>
S1	01	joe@bigu	3.5
S1	02	joe@bigu	3.6
S2	01	mary@bigu	3.8
S2	03	mary@bigu	3.5

```

graph TD
    StdSSN --> OfferNo
    StdSSN --> Email
    StdSSN --> EnrGrade
    OfferNo --> EnrGrade
    Email --> EnrGrade
  
```

UnivTable4-1 (OfferNo, StdSSN, EnrGrade)  
FOREIGN KEY (StdSSN) REFERENCES UnivTable4-2  
UnivTable4-2 (StdSSN, Email)

**FIGURA 7.5**  
Filas de ejemplo,  
diagrama de  
dependencias y tablas  
normalizadas de  
*UnivTable5*

<b>UnivTable5</b>			
<u>StdSSN</u>	<u>AdvisorNo</u>	<u>Major</u>	<u>Status</u>
S1	A1	IS	COMPLETED
S1	A2	FIN	PENDING
S2	A1	IS	PENDING
S2	A3	FIN	COMPLETED

```

graph TD
    AdvisorNo --> StdSSN
    AdvisorNo --> Major
    AdvisorNo --> Status
    StdSSN --> Major
    Major --> Status
  
```

UnivTable5-1 (AdvisorNo, StdSSN, Status)  
FOREIGN KEY (AdvisorNo) REFERENCES UnivTable5-2  
UnivTable5-2 (AdvisorNo, Major)

- 3NF: *UnivTable5* está en 3NF ya que *Major* es una columna llave. *Status* es la única columna que no forma parte de la llave. *UnivTable5* está en 3NF debido a que *Status* depende de las llaves candidatas en su totalidad ( $<\text{StdSSN}, \text{AdvisorNo}>$  y  $<\text{StdSSN}, \text{Major}>$ ).
- BCNF: El diagrama de dependencias (figura 7.5) muestra que *AdvisorNo* es un determinante pero no una llave candidata por sí misma. Por ende, *UnivTable5* no está en BCNF. Para eliminar la redundancia, debe separar *UnivTable5* en dos tablas, tal como se muestra en la figura 7.5.

Estos ejemplos demuestran dos puntos acerca de la normalización. Primero, las tablas con llaves candidatas compuestas y múltiples son difíciles de analizar. Necesita estudiar cuidadosamente las dependencias en cada uno de los ejemplos (figuras 7.4 y 7.5) para entender las conclusiones acerca de las violaciones cometidas contra la forma normal. Segundo, la mayoría de las tablas en 3NF (incluso las que tienen múltiples llaves candidatas compuestas) están también en BCNF. Los ejemplos de las figuras 7.4 y 7.5 se construyeron con el propósito de ilustrar la diferencia entre 3NF y BCNF. La importancia de BCNF es que es una definición más sencilla y se puede aplicar en el procedimiento descrito en la siguiente sección.

### 7.2.4 Procedimiento de síntesis simple

El procedimiento de síntesis simple se puede usar para generar tablas que satisfagan BCNF, comenzando con una lista de dependencias funcionales. La palabra *síntesis* significa que las dependencias funcionales individuales se combinan para construir tablas. Este uso es semejante al de otras disciplinas como la música, en donde la síntesis involucra la combinación de sonidos individuales para construir unidades más grandes como las melodías, sinfonías, etcétera.

La figura 7.6 muestra los pasos del procedimiento de la síntesis simple. Los primeros dos pasos eliminan la redundancia removiendo las columnas ajenas y FD derivadas. Los últimos tres pasos generan las tablas para las colecciones de las FD. Las tablas generadas en los últimos tres pasos pueden no ser correctas si no se eliminan las FD redundantes.

#### *Aplicación del procedimiento de la síntesis simple*

Para comprender este procedimiento, puede aplicarlo a las FD de la tabla de la base de datos de la universidad (tabla 7.2). En el primer paso, no existen columnas ajenas en los determinantes. Para demostrar que existe una columna ajena, suponga que existe la FD  $\text{StdSSN}, \text{StdCity} \rightarrow \text{StdClass}$ . En esta FD, si se elimina *StdCity* del lado izquierdo, aún se conservaría la FD  $\text{StdSSN} \rightarrow \text{StdClass}$ . La columna *StdCity* es redundante en la FD y debe ser eliminada.

Para aplicar el segundo paso, usted necesita saber cómo derivar las FD a partir de otras FD. Aunque existen varias formas de derivar las FD, la forma más popular es mediante la ley de la transitividad tal como se estableció en los comentarios sobre 3NF (sección 7.2.2). Para nuestros propósitos, eliminaremos las FD transitivas derivadas en el paso 2. Para obtener más detalles acerca de otras formas de derivar las FD, debe consultar las referencias listadas al final del capítulo.

En el segundo paso, la FD  $\text{OfferNo} \rightarrow \text{CrsDesc}$  es una dependencia transitiva, ya que  $\text{OfferNo} \rightarrow \text{CourseNo}$  y  $\text{CourseNo} \rightarrow \text{CrsDesc}$  implica  $\text{OfferNo} \rightarrow \text{CrsDesc}$ . Por lo tanto, debe eliminar esta dependencia de la lista de FD.

**FIGURA 7.6**  
Pasos del procedimiento de síntesis simple

1. Elimine las columnas ajenas que están a mano izquierda de las FD.
2. Elimine las FD derivadas de la lista de FD.
3. Acomode las FD en grupos en los que cada uno tenga el mismo determinante.
4. Para cada grupo FD, haga una tabla en donde el determinante sea la llave primaria.
5. Combine las tablas en las que una tabla contenga todas las columnas de la otra tabla.
  - 5.1. Elija la llave primaria de una de las tablas separadas como la llave primaria de la nueva tabla combinada.
  - 5.2. Defina las restricciones de exclusividad para el resto de las llaves primarias que no fueron designadas como la llave primaria de la tabla nueva.

En el tercer paso, debe agrupar las FD con base en el determinante. A partir de la tabla 7.2, puede crear los siguientes grupos FD:

- $StdSSN \rightarrow StdCity, StdClass$
- $OfferNo \rightarrow OffTerm, OffYear, CourseNo$
- $CourseNo \rightarrow CrsDesc$
- $StdSSN, OfferNo \rightarrow EnrGrade$

En el cuarto paso, reemplace cada grupo FD con una tabla que tenga el determinante en común, como la llave primaria. Así, tendrá cuatro tablas BCNF resultantes tal como se muestra más abajo. Debe agregar los nombres de las tablas para completar el ciclo.

**Student**(StdSSN, StdCity, StdClass)  
**Offering**(OfferNo, OffTerm, OffYear, CourseNo)  
**Course**(CourseNo, CrsDesc)  
**Enrollment**(StdSSN, OfferNo, EnrGrade)

Después de definir las tablas, debe agregar las restricciones de integridad referencial para conectarlas. Para detectar la necesidad de una restricción de integridad referencial, debe buscar una llave primaria de una tabla que aparezca en otras. Por ejemplo, *CourseNo* es la llave primaria de *Course* pero también aparece en *Offering*. Por lo tanto, debe definir una restricción de integridad referencial indicando que *Offering.CourseNo* hace referencia a *Course.CourseNo*. Abajo se repiten las tablas agregando las restricciones de integridad referencial.

**Student**(StdSSN, StdCity, StdClass)  
**Offering**(OfferNo, OffTerm, OffYear, CourseNo)  
    FOREIGN KEY (CourseNo) REFERENCES Course  
**Course**(CourseNo, CrsDesc)  
**Enrollment**(StdSSN, OfferNo, EnrGrade)  
    FOREIGN KEY (StdSSN) REFERENCES Student  
    FOREIGN KEY (OfferNo) REFERENCES Offering

El quinto paso no es necesario, ya que las FD para este problema son sencillas. El quinto paso es necesario cuando existen varias llaves candidatas para una tabla. Por ejemplo, si se agrega *Email* como una columna, entonces se deben agregar a la lista las FD *Email*  $\rightarrow$  *StdSSN* y *StdSSN*  $\rightarrow$  *Email*. Observe que las FD *Email*  $\rightarrow$  *StdCity*, *StdClass* no se deben agregar a la lista dado que estas FD se pueden derivar transitoriamente de otras FD. Como resultado del paso 3, se agrega otro grupo de FD. En el paso 4, se agrega una tabla nueva (*Student2*) con *Email* como la llave primaria. Debido a que la tabla *Student* contiene las columnas de la tabla *Student2*, las tablas (*Student* y *Student2*) se combinan en el paso 5. Como llave primaria se escoge a una de las llaves candidatas (*StdSSN* o *Email*). Dado que se escogió a *Email* como la llave primaria, se define una restricción de exclusividad para *StdSSN*.

*Email*  $\rightarrow$  *StdSSN*  
*StdSSN*  $\rightarrow$  *Email*

**Student2**(Email, StdSSN, StdCity, StdClass)  
    UNIQUE(StdSSN)

Como lo demuestra este ejemplo adicional, múltiples llaves candidatas no violan la BCNF. El quinto paso del procedimiento de la síntesis simple crea tablas con múltiples llaves candidatas ya que combina las tablas. Múltiples llaves candidatas tampoco violan 3NF. No hay razón para dividir una tabla sólo por tener múltiples llaves candidatas. La división de una tabla con

### múltiples llaves candidatas

un malentendido general de los nuevos desarrolladores de bases de datos es que una tabla con múltiples llaves candidatas viola el BCNF. Estas llaves no violan el BCNF o 3NF. Por ende, no debe dividir una tabla sólo porque tiene múltiples llaves candidatas.

múltiples llaves candidatas puede hacer más lento el desempeño de las consultas debido a los enlaces adicionales.

Usted puede usar el procedimiento de síntesis simple para analizar las estructuras de dependencia simples. La mayoría de las tablas que resultan de la conversión de un ERD deberán tener estructuras de dependencia simple debido a que el proceso de modelado de datos ya ha hecho mucho del proceso de normalización. La mayor parte de las tablas deberán estar casi normalizadas después del proceso de conversión.

Para estructuras de dependencia complejas, debe emplear una herramienta comercial de diseño para llevar a cabo la normalización. Para facilitar el procedimiento de síntesis se han omitido algunos detalles. De manera particular, el paso 2 de preferencia se debe involucrar porque hay más formas de derivar las dependencias que la transitividad. Incluso puede ser difícil revisar la transitividad cuando existen muchas columnas. Los detalles completos se pueden encontrar en la sección de referencias citadas al final del capítulo. Aún si se comprenden los detalles complejos, el paso 2 no puede hacerse de forma manual para las estructuras de dependencia complejas. Para ellas, usted necesita emplear una herramienta CASE, aun cuando sea un diseñador experimentado de bases de datos.

#### *Otro ejemplo del uso del procedimiento de síntesis simple*

Para obtener más experiencia con el procedimiento de síntesis simple, debe comprender otro ejemplo. Este ejemplo describe una base de datos que permite rastrear revisiones de los documentos que se presentan en una conferencia académica. Los autores considerados presentan documentos para revisión y posible aceptación en el proceso de publicación de la conferencia. Aquí hay más detalles sobre autores, documentos, revisiones y revisores:

- La información del autor incluye el número único de autor, el nombre, la dirección postal y la única, aunque opcional, dirección de correo electrónico.
- La información del documento abarca el autor principal, un número único de documento, el título, el resumen y el estado de la revisión (pendiente, aceptado, rechazado).
- La información del revisor consiste en el número único de revisor, el nombre, la dirección postal y la única, aunque opcional, dirección de correo electrónico.
- Una revisión completa implica el número de revisor, la fecha, el número de documento, los comentarios del presidente del programa y las clasificaciones (general, originalidad, corrección, estilo y relevancia). La combinación del número de revisor y el número de documento identifica a la revisión.

Antes de comenzar con el procedimiento, es necesario identificar las FD en el problema. La siguiente es una lista de FD para el problema:

$$\begin{aligned}AuthNo \rightarrow AuthName, AuthEmail, AuthAddress \\AuthEmail \rightarrow AuthNo \\PaperNo \rightarrow Primary-AuthNo, Title, Abstract, Status \\RevNo \rightarrow RevName, RevEmail, RevAddress \\RevEmail \rightarrow RevNo \\RevNo, PaperNo \rightarrow Auth-Comm, Prog-Comm, Date, Rating1, Rating2, Rating3, \\Rating4, Rating5\end{aligned}$$

Debido a que LHS es mínimo en cada FD, el primer paso está terminado. El segundo paso no es necesario porque no hay dependencias transitivas. Observe que la FD  $AuthEmail \rightarrow AuthName, AuthAddress$ , y  $RevEmail \rightarrow RevName, RevAddress$  puede ser derivada de manera transitiva. Si cualquiera de estas FD fueran parte de un listado original, deberán ser eliminadas. Se debe definir una tabla para cada uno de los seis grupos FD. En el último paso, se combinan los grupos FD con  $AuthNo$  y  $AuthEmail$ , y  $RevNo$  y  $RevEmail$  como determinantes. Además, debe añadir las restricciones únicas para  $AuthEmail$  y  $RevEmail$  porque dichas columnas no fueron seleccionadas como las claves primarias de las nuevas tablas.

```

Author(AuthNo, AuthName, AuthEmail, AuthAddress)
    UNIQUE (AuthEmail)
Paper(PaperNo, Primary-Auth, Title, Abstract, Status)
    FOREIGN KEY (Primary-Auth) REFERENCES Author
Reviewer(RevNo, RevName, RevEmail, RevAddress)
    UNIQUE (RevEmail)
Review(PaperNo, RevNo, Auth-Comm, Prog-Comm, Date, Rating1, Rating2,
    Rating3, Rating4, Rating5)
    FOREIGN KEY (PaperNo) REFERENCES Paper
    FOREIGN KEY (RevNo) REFERENCES Reviewer

```

## 7.3 Refinamiento de las relaciones M-way

---

Más allá de la BCNF, una cuestión pendiente es el análisis de las relaciones M-way. Recuerde que las relaciones M-way están representadas por tipos de entidad asociativa en la notación de ERD de pata de cuervo. En el proceso de conversión, un tipo de entidad asociativa se convierte en una tabla con una llave primaria combinada que consiste en tres o más componentes. El concepto de independencia, que radica en 4NF, es una importante herramienta para el análisis de las relaciones M-way. Por medio del concepto de independencia usted puede ver que una relación M-way debe ser dividida en dos o más relaciones binarias para evitar la redundancia. En el capítulo 12 usted hará uso de formularios para analizar la necesidad de relaciones M-way. Las siguientes secciones describen el concepto de independencia de relación y 4NF.

### 7.3.1 Independencia de relación

Antes de estudiar la manera en que la independencia influye en el diseño de la base de datos, comentaremos el significado de independencia en estadística. Dos variables son estadísticamente independientes si al conocer algo de alguna de las variables no significa nada para la otra. De forma más precisa, dos variables son independientes si la probabilidad de ambas variables (probabilidad conjunta) puede derivarse de la probabilidad de cada variable por sí misma. Por ejemplo, una variable puede ser la edad de una piedra y otra variable puede tener la edad de la persona que sostiene la piedra. Debido a que la edad de la roca y la edad de la persona que la sostiene no están relacionadas, estas variables se consideran independientes. No obstante, la edad de la persona y su estado civil sí están relacionadas. Conocer la edad de una persona nos da indicios sobre la probabilidad de que sea soltera, casada o divorciada. Si las dos variables son independientes, es redundante almacenar datos acerca de la manera en que se relacionan. Usted puede usar probabilidades acerca de las variables individuales para derivar las probabilidades conjuntas.

**independencia de la relación**  
relación que puede derivarse de dos relaciones independientes.

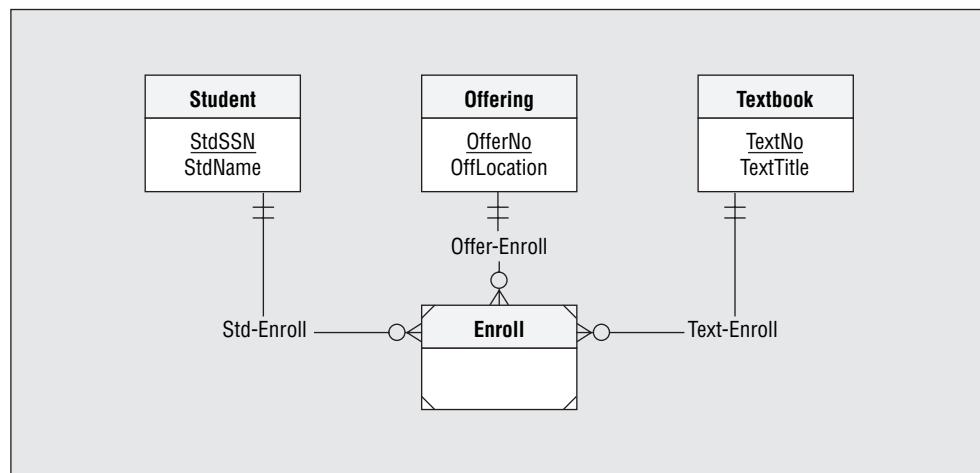
El concepto de *independencia de relación* es similar al de independencia estadística. Si dos relaciones son independientes (es decir, que no están relacionadas entre sí), es redundante almacenar datos acerca de una tercera relación. Ésta puede derivarse al combinar las dos relaciones esenciales por medio de una operación de enlace. Si usted almacena la relación derivada pueden resultar anomalías en la modificación. Así, la idea esencial de la independencia de relación no es almacenar las relaciones que pueden derivarse al enlazarse con otras relaciones (independientes).

#### *Ejemplo de independencia de relación*

Para aclarar la independencia de relación, considere el tipo de entidad asociativa *Enroll* (figura 7.7) que representa una relación de tres vías entre los alumnos, la oferta y los libros de texto. El tipo de entidad *Enroll* convierte la tabla *Enroll* (tabla 7.6) que consiste únicamente en una llave primaria combinada: *StdSSN*, *OfferNo* y *TextNo*.

La cuestión del diseño es si la tabla *Enroll* tiene redundancias. En caso de haber redundancias, pueden presentarse anomalías de modificación. La tabla *Enroll* está en BCNF, de manera que no hay anomalías debido a dependencias funcionales. Sin embargo, el concepto de

**FIGURA 7.7**  
Ejemplo de relación  
M-way



**TABLA 7.6**  
Filas de muestra de  
la tabla *Enroll*

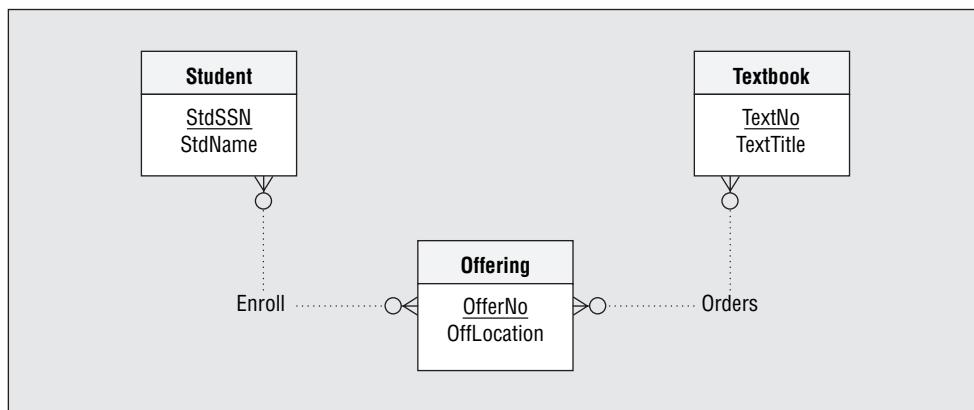
StdSSN	OfferNo	TextNo
S1	O1	T1
S1	O2	T2
S1	O1	T2
S1	O2	T1

independencia nos lleva al descubrimiento de las redundancias. La tabla *Enroll* puede dividirse en tres combinaciones de columnas que representan tres relaciones binarias: *StdSSN-OfferNo* representa la relación entre el alumno y la oferta, *OfferNo-TextNo* equivale a la relación entre la oferta y el libro de texto, y *StdSSN-TextNo* indica la relación que existe entre el alumno y el libro de texto. Si cualquiera de las relaciones binarias pudiera derivarse de las otras dos, entonces, se dice que hay redundancia.

- La relación entre los alumnos y las ofertas (*StdSSN-OfferNo*) no puede derivarse de las otras dos relaciones. Por ejemplo, suponga que el libro de texto T1 se utiliza en dos ofertas, O1 y O2, y por dos alumnos, S1 y S2. Conociendo estos hechos, usted no conoce la relación que hay entre los alumnos y las ofertas. Por ejemplo, S1 podría participar en O1 o quizás en O2.
- De la misma manera, la relación entre la oferta y los libros de texto (*OfferNo-TextNo*) no puede derivarse. La selección de un profesor por cierta colección de libros de texto no puede derivarse al conocer quién participa en una oferta y qué libros de texto son los que utiliza el alumno.
- No obstante, la relación entre alumnos y libros de texto (*StdSSN-TextNo*) sí puede derivarse de las otras dos relaciones. Por ejemplo, si el estudiante S1 participa en la oferta O1 y O1 emplea el libro de texto T1, entonces, se puede concluir que el alumno S1 utiliza el libro de texto T1 en la oferta O1. Debido a que las relaciones *Student-Offering* y *Offering-Textbook* son independientes, usted puede conocer los libros de texto empleados por un alumno sin tener que almacenar las instancias de las relaciones.

Por su independencia, la tabla *Enroll* y el tipo de entidad asociativa relacionado tienen redundancia. Para eliminar dicha redundancia, reemplace el tipo de entidad *Enroll* con dos relaciones binarias (figura 7.8). Cada relación binaria se convierte en una tabla, como se ilustra en las tablas 7.7 y 7.8. Las tablas *Enroll* y *Orders* no tienen redundancia alguna. Por ejemplo, para eliminar la participación de un alumno en una oferta (digamos, S1 en O1), debe eliminarse una sola fila de la tabla 7.7. En contraste, deben eliminarse dos filas en la tabla 7.6.

**FIGURA 7.8**  
Ejemplo de relaciones descompostas



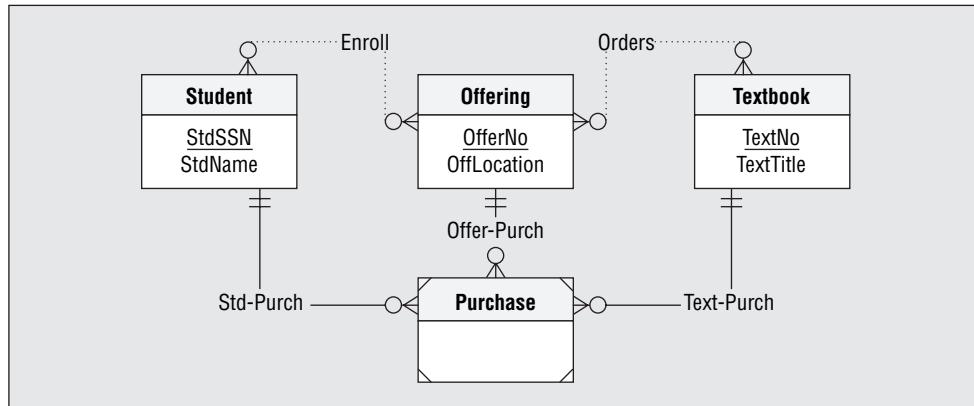
**TABLA 7.7**  
Ejemplo de filas de la tabla binaria *Enroll*

StdSSN	OfferNo
S1	O1
S1	O2

**TABLA 7.8**  
Ejemplo de filas de la tabla binaria *Orders*

OfferNo	TextNo
O1	T1
O1	T2
O2	T1
O2	T2

**FIGURA 7.9**  
Ejemplo de relaciones binarias y M-way



Si las consideraciones cambian ligeramente, se puede establecer un argumento para el tipo de entidad asociativa que representa una relación de tres vías. Suponga que la librería quiere registrar las compras de los libros de texto por ofertas y por alumnos y con ello estimar la demanda de libros de texto. Entonces, la relación entre libros de texto y alumnos ya no es más independiente de las otras dos relaciones. Incluso si un alumno está participando en una oferta y ésta utiliza un libro de texto, el alumno podría no comprar el libro de la oferta (quizá lo pida prestado). En esta situación, no hay independencia y se necesita una relación de tres vías. Además de las relaciones M-N que aparecen en la figura 7.8, deberá existir un nuevo tipo de entidad asociativa y tres relaciones 1-M, como se muestra en la figura 7.9. Se necesita la relación *Enroll* para registrar las selecciones de los alumnos en cuanto a las ofertas, y la relación *Orders* para registrar las selecciones de los profesores en lo que se refiere a libros de texto. El tipo de entidad *Purchase* registra las compras de los libros de texto hechas por los estudiantes en una oferta. Sin embargo, una compra no puede conocerse de las otras relaciones.

### 7.3.2 Dependencias multivaluadas y cuarta forma normal

#### definición de MVD

la dependencia multivaluada (MVD, por sus siglas en inglés)  $A \rightarrow\rightarrow B \mid C$  (se lee  $A$  multidetermina a  $B$  o  $C$ ) significa que:

- Un valor proporcionado a  $A$  está asociado con una colección de valores de  $B$  y  $C$ , y
- $B$  y  $C$  son independientes dadas las relaciones entre  $A$  y  $B$  y  $A$  y  $C$ .

En la terminología de bases de datos relacionales, una relación que puede derivarse de otras relaciones se conoce como dependencia multivaluada (MVD, por sus siglas en inglés). Una MVD implica tres columnas, tal y como se describe en la definición que aparece al margen. Como se comentó sobre la independencia de relaciones, las tres columnas implican una llave primaria combinada de una tabla asociativa. Las relaciones no esenciales o derivadas implican a las columnas  $B$  y  $C$ . La definición establece que la relación no esencial (que involucra a las columnas  $B$  y  $C$ ) puede derivarse de las relaciones  $A-B$  y  $A-C$ . La palabra *multivaluada* significa que  $A$  puede asociarse con una colección de valores de  $B$  y  $C$ , no sólo con valores sencillos como sucede en una dependencia funcional.

Las MVD pueden ocasionar redundancias por la independencia que hay entre las columnas. Usted puede ver la redundancia al usar una tabla para marcar una MVD, como se muestra en la figura 7.10. Si los dos registros que están sobre la línea existen y la MVD  $A \rightarrow\rightarrow B \mid C$  es verdadera, entonces existirán ambos registros debajo de la línea. Los dos registros debajo de la línea existirán porque la relación entre  $B$  y  $C$  puede derivarse de las relaciones  $A-B$  y  $A-C$ . En la figura 7.10, el valor de  $A1$  está asociado con dos valores de  $B$  ( $B1$  y  $B2$ ) y dos valores de  $C$  ( $C1$  y  $C2$ ). Debido a la independencia, el valor  $A1$  será asociado con cada combinación de los valores  $B$  y  $C$  que estén relacionados con él. Los dos registros que están debajo de la línea son redundantes porque pueden derivarse.

Para aplicar este concepto a la tabla *Enroll*, considere la posible MVD  $OfferNo \rightarrow\rightarrow StdSSN \mid TextNo$ . En los dos primeros registros de la figura 7.11, la oferta  $O1$  está asociada con los estudiantes  $S1$  y  $S2$  y con los libros de texto  $T1$  y  $T2$ . Si la MVD es verdadera, entonces, los dos registros que están debajo de la línea existirán. Los dos últimos registros no necesitan almacenarse si usted conoce los dos registros y la MVD existe.

Las MVD son generalizaciones de dependencias funcionales (FD). Cada FD es una MVD, pero no toda MVD es una FD. Una MVD en la que el valor de  $A$  se ve asociado con sólo un valor de  $B$  y un valor de  $C$ , también es una FD. En esta sección nos interesan únicamente las MVD que no son también FD. Una MVD que no es también una FD se conoce como MVD no trivial.

#### definición 4NF

una tabla se encuentra en 4NF cuando no contiene ningún MVD no trivial (MVD que también son FD).

#### Cuarta forma normal (4NF)

La cuarta forma normal (4NF) prohíbe las redundancias causadas por las dependencias multivaluadas. Por ejemplo, la tabla *Enroll(StdSSN, OfferNo, TextNo)* (tabla 7.6) no es 4NF si la MVD  $OfferNo \rightarrow\rightarrow StdSSN \mid TextNo$  existe. Para eliminar la MVD, divida la tabla M-way *Enroll* en tablas binarias *Enroll* (tabla 7.7) y *Orders* (tabla 7.8).

**FIGURA 7.10**

Tabla de representación de una MVD

A	B	C
A1	B1	C1
<u>A1</u>	<u>B2</u>	<u>C2</u>
A1	B2	C1
A1	B1	C2

**FIGURA 7.11**

Representación de la MVD en la tabla *Enroll*

OfferNo	StdSSN	TextNo
O1	S1	T1
<u>O1</u>	<u>S2</u>	<u>T2</u>
O1	S2	T1
O1	S1	T2

Las ideas de MVD y 4NF son algo difíciles de entender. Las ideas son más fáciles de entender si usted piensa en una MVD como una relación que puede derivarse por otras relaciones como consecuencia de la independencia. El capítulo 12 presenta otra forma de razonar acerca de las relaciones M-way por medio de los patrones que hay en los formularios de captura de datos.

## 7.4 Formas normales de alto nivel

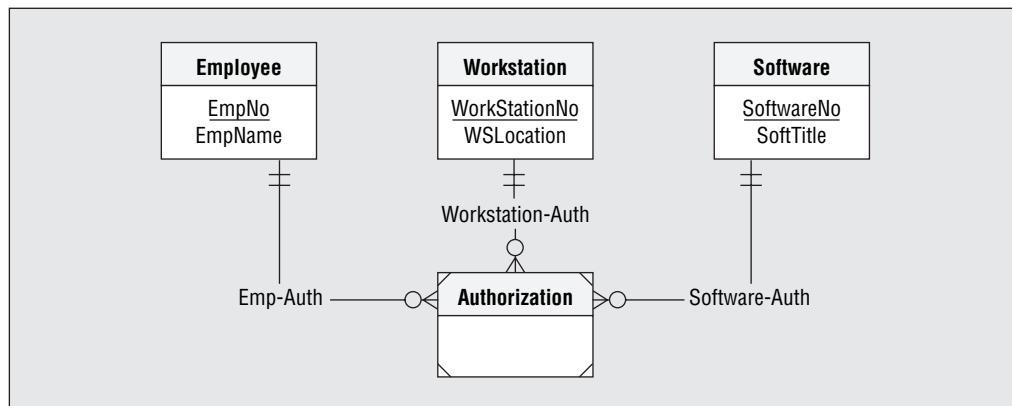
La historia de la normalización no termina con la 4NF. Se han propuesto otras formas normales, pero no se ha demostrado su practicidad. Esta sección describe con brevedad dos formas de alto nivel para completar sus conocimientos de normalización.

### 7.4.1 Quinta forma normal

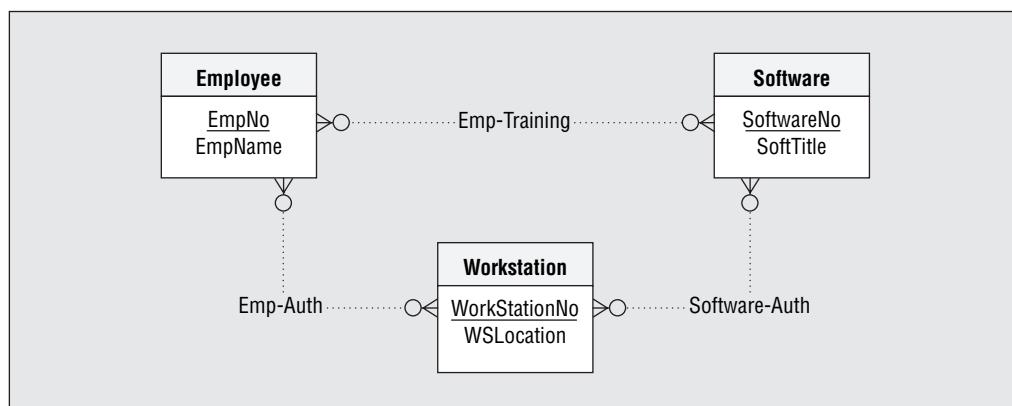
La quinta forma normal (5NF) aplica a las relaciones M-way como lo hace 4NF. A diferencia de 4NF, 5NF implica situaciones en donde una relación de tres vías debe ser reemplazada con tres relaciones binarias, y no con dos como sucede con el caso de 4NF. Como son raras las situaciones en las que aplica la 5NF (en contraposición con 4NF), por lo general a 5NF no se le considera como una forma normal práctica. La comprensión de los detalles de 5NF implica mucha inversión intelectual, pero la ganancia por el tiempo de estudio rara vez se aplica.

El ejemplo en la figura 7.12 demuestra una situación en la que podría aplicar una 5NF. El tipo de entidad de *Authorization* representa combinaciones autorizadas de empleados, estaciones de trabajo y software. Este tipo de entidad asociativa tiene redundancia porque puede dividirse en tres relaciones binarias, como se muestra en la figura 7.13. Si usted conoce empleados

**FIGURA 7.12**  
Tipo de entidad asociativa



**FIGURA 7.13**  
Reemplazo del tipo de entidad asociativa con tres relaciones binarias



autorizados para emplear estaciones de trabajo, licencias de software para estaciones de trabajo y empleados capacitados para usar software, entonces usted conoce las combinaciones válidas de empleados, estaciones de trabajo y software. Así, es necesario recordar las tres combinaciones binarias (empleado-estación de trabajo, software-estación de trabajo, y empleado-software), no la combinación de tres vías para el empleado, estación de trabajo y software.

Si la situación mostrada en la figura 7.13 es real, entonces es debatible. Por ejemplo, si el software tiene licencia para servidores en lugar de estaciones de trabajo, puede no ser necesaria la relación *Software-Auth*. Incluso a pesar de que es posible representar situaciones en las que aplica la 5NF, dichas situaciones podrían no existir en organizaciones reales.

### 7.4.2 Forma normal de Llave de dominio

Después de leer acerca de tantas formas normales, tal vez se haga preguntas como: “¿dónde termina todo esto?”, o “¿existe una forma normal definitiva?”. Afortunadamente, la respuesta a esta última pregunta es sí. En un documento de 1981, el doctor Ronald Fagin propuso una forma normal de llave de dominio (DKNF) como una forma normal definitiva. En DKNF, dominio se refiere a un tipo de dato: una serie de valores con operaciones permisibles. Una serie de valores se define por el tipo de valores (por ejemplo, números enteros *versus* números de punto flotante) y las reglas de integridad acerca de los valores (por ejemplo, valores superiores a 21). Llave se refiere a la propiedad de exclusividad de las llaves candidatas. Una tabla está en DKNF si cada restricción de una tabla puede derivarse a partir de llaves y dominios. Una tabla en DKNF no puede tener anomalías de modificación.

Desafortunadamente DKNF permanece como una forma ideal en lugar de una forma normal práctica. No hay algoritmo conocido que convierta una tabla en un DKNF. Además, no se conoce qué tablas pueden convertirse a DKNF. En su calidad de ideal, usted debe intentar definir tablas en las que resultan la mayor parte de las restricciones a partir de llaves y dominios. Este tipo de restricciones son fáciles de evaluar y de comprender.

## 7.5 Cuestiones prácticas acerca de normalización

---

**ventajas de normalización como herramienta de refinamiento**  
utilice la normalización para eliminar las redundancias después de la conversión de un ERD a una tabla de diseño en lugar de una herramienta inicial de diseño porque:

- Es más fácil traducir los requisitos en un ERD en lugar de listas de FD.
- Son menos las FD por especificar porque la mayor parte de éstas se derivan de llaves primarias.
- Son menos las tablas a dividir porque la normalización se llevó a cabo de manera intuitiva durante el desarrollo del ERD.
- Es más fácil identificar las relaciones, especialmente las relaciones M-N sin atributos.

Después de leer todo esto, debe estar bastante familiarizado con las herramientas de diseño de bases de datos relacionales. Antes de estar listo para usar estas herramientas, le serán útiles algunos consejos prácticos. Esta sección expone el papel de la normalización en el proceso de desarrollo de bases de datos y la importancia de considerar detenidamente los objetivos de eliminar anomalías de modificación.

### 7.5.1 Función de la normalización en el proceso de desarrollo de base de datos

Hay dos formas distintas de usar la normalización en el proceso de desarrollo de la base de datos: (1) como una herramienta de refinamiento, o (2) como una herramienta de diseño inicial. En la metodología de refinamiento usted lleva a cabo un modelado de datos conceptuales por medio del modelo de entidad-relación y transforma el ERD en tablas por medio de las reglas de conversión. Después, aplica las técnicas de normalización para analizar cada una de las tablas: identifica FD, utiliza el procedimiento de síntesis simple para eliminar las redundancias y analiza que la tabla sea independiente si es que presenta una relación M-way. Ya que la llave primaria determina las otras columnas de la tabla, usted únicamente debe identificar las FD en las que la llave primaria no es el LHS.

En la metodología de diseño inicial usted emplea las técnicas de normalización en un modelado de datos conceptuales. En lugar de dibujar un ERD, usted identifica las dependencias funcionales y aplica un procedimiento de normalización semejante al procedimiento de síntesis simple. Después de definir las tablas, usted identifica las restricciones de integridad referencial y construye un diagrama del modelo referencial como el que está disponible en Microsoft Access. De ser necesario, se puede generar un ERD a partir del diagrama de bases de datos relacionales.

Este libro favorece claramente el uso de la normalización como herramienta de refinamiento y no como herramienta de diseño inicial. Por medio del desarrollo de un ERD, usted agrupa de forma intuitiva los campos relacionados. Mucha de la normalización se realiza de manera informal sin el proceso tedioso de registrar las dependencias funcionales. Como herramienta de refinamiento, hay menos FD por especificar y menos normalización por llevar a cabo. La aplicación de la normalización asegura que no se han pasado por alto las llaves candidatas ni las redundancias.

Otra razón para favorecer la metodología de refinamiento es que se pueden pasar por alto las relaciones cuando se usa la normalización como la metodología de diseño inicial. Las relaciones 1-M deben identificarse en la dirección hija-a-madre. Para los modeladores de datos que no tienen mucha experiencia, la identificación de las relaciones es más sencilla cuando se consideran ambos lados de una relación. Para una relación M-N sin atributos, no habrá dependencia funcional alguna que muestre la necesidad de una tabla. Por ejemplo, en un diseño sobre libros de texto y ofertas para el curso, si la relación entre ellos no tiene atributos, no hay dependencias funcionales que relacionen a libros de texto con ofertas del curso.<sup>6</sup> Al dibujar un ERD, sin embargo, la necesidad de una relación M-N se hace evidente.

### 7.5.2 Análisis del objetivo de normalización

Como criterio de diseño, evitar anomalías de modificación se desvía hacia los cambios en la base de datos. Como usted ha visto, la eliminación de las anomalías normalmente resulta en una base de datos con muchas tablas. Un diseño que contenga muchas tablas permite que la base de datos tenga mayor facilidad para cambiar, pero más dificultades con las consultas. Si la base de datos se emplea de forma predominante para consultas, la eliminación de las anomalías de modificación no se presenta como la mejor meta de diseño. El capítulo 16 describe las bases de datos para el apoyo de decisiones en las que el uso primordial es el de consultas en lugar de modificaciones. En esta situación, un diseño que no está plenamente normalizado puede ser el más adecuado. La desnormalización es el proceso de combinar tablas para que sea más fácil realizar consultas. Además, las metas de diseño físico pueden tener conflicto con las metas lógicas de diseño. El capítulo 8 describe las metas del diseño físico de base de datos y el uso de la desnormalización como una técnica para mejorar el desempeño de las consultas.

Otro momento en el que se puede considerar la desnormalización es cuando una FD no es importante. El ejemplo clásico contiene las FD *Zip → City, State* en una tabla de clientes en donde *City* significa ciudad de la oficina postal. En algunas bases de datos, puede no ser importante conservar estas dependencias. Si no existe necesidad de manipular los códigos postales de manera independiente de los clientes, se pueden ignorar con toda seguridad las FD. No obstante, hay bases de datos en las que es importante mantener una tabla de códigos postales de forma independiente a la información del cliente. Por ejemplo, si un distribuidor tiene negocios en muchos estados y países, una tabla de códigos postales es útil para registrar las cuotas fiscales de las ventas.<sup>7</sup> Si usted desea ignorar una FD en el proceso de normalización, debe tener en cuenta que ésta existe pero que no le llevará a ninguna anomalía significativa. Proceda con cautela: de ser ignoradas, *la mayoría* de las FD llevan a anomalías.

**uso de la desnormalización**  
considere la violación de BCNF como objetivo de diseño para una tabla cuando:

- Una FD no es importante para reforzar una restricción de llave candidata.
- Una base de datos se usa principalmente para consultas.
- El desempeño de la consulta requiere de menos tablas para reducir el número de operaciones de enlace (*join*).

---

## Reflexión final

Este capítulo describió la manera en la que las redundancias podrían dificultar el cambio en una tabla debido a que causan anomalías. Evitar las anomalías es el objetivo de las técnicas de normalización. Debe enlistar las dependencias funcionales (FD) como prerequisito para la normalización de una tabla. Este capítulo ha descrito tres formas normales (2NF, 3NF y BCNF)

<sup>6</sup> Se puede escribir una FD en el lado derecho nulo para representar las relaciones M-N. La FD para la relación oferta-libro de texto puede expresarse como *TextId, OfferNo → ∅*. Sin embargo, este tipo de FD es difícil de establecer. Es mucho más fácil definir una relación M-N.

<sup>7</sup> Un antiguo alumno de bases de datos hizo este comentario acerca de la base de datos de un gran distribuidor de computadoras.

con base en las dependencias funcionales. Se presentó el procedimiento de síntesis simple para analizar las dependencias funcionales y producir las tablas en BCNF. El proporcionar una lista completa de FD es la parte más importante del proceso de normalización. Incluso si usted no entiende las formas normales, puede adquirir una herramienta CASE para llevar a cabo la normalización. Sin embargo, las herramientas CASE no tienen la capacidad de proporcionar una lista completa de FD.

Este capítulo también describió una metodología para analizar las relaciones M-way (representadas por los tipos de entidad asociativa) por medio del concepto de independencia. Si dos relaciones son independientes, la tercera relación se derivará de ellas. No hay necesidad de almacenar la tercera relación. El concepto de independencia es equivalente a la dependencia multivaluada. 4NF no permite la redundancia causada por dependencias multivaluadas.

Este capítulo y los que tratan sobre el modelado de datos (capítulos 5 y 6) enfatizaron las habilidades fundamentales para el desarrollo de bases de datos. Después de que el modelado de datos y la normalización están completos, usted está listo para implementar el diseño, usualmente por medio de los DBMS relacionales. El capítulo 8 describe los conceptos y las prácticas del diseño físico de bases de datos para facilitar su labor de implementación en DBMS relacionales.

## Revisión de conceptos

- Las redundancias de una tabla causan anomalías de modificación.
- Anomalías de modificación: efectos colaterales inesperados cuando se inserta, actualiza o elimina.
- Dependencias funcionales: una restricción de valor neutral similar al de la llave primaria.
- 2NF: columnas sin llaves que dependen de una llave entera y no de una subserie de llaves.
- 3NF: columnas sin llaves que dependen exclusivamente de la llave y no de otras columnas sin llaves.
- BCNF: cada determinante es una llave candidata.
- Procedimiento de síntesis simple: analiza FD y produce tablas en BCNF.
- Use el procedimiento de síntesis simple para analizar las estructuras de dependencias simples.
- Use software de diseño comercial para analizar las estructuras de dependencias complejas.
- Use la independencia de relaciones como un criterio para dividir las relaciones M-way en relaciones más pequeñas.
- MVD: asociación con colecciones de valores e independencia entre las columnas.
- Las MVD causan redundancia porque los registros pueden derivarse por medio de la independencia.
- 4NF: no hay redundancias ocasionadas por MVD.
- Use las técnicas de desnormalización como una herramienta de refinamiento en lugar de emplearlas como una herramienta de diseño inicial.
- Desnormalice una tabla si las FD no ocasionan anomalías de modificación.

## Preguntas

1. ¿Qué es una anomalía de inserción?
2. ¿Qué es una anomalía de actualización?
3. ¿Qué es una anomalía de eliminación?
4. ¿Cuál es la causa de las anomalías de modificación?
5. ¿Qué es una dependencia funcional?
6. ¿En qué se parece una dependencia funcional a una llave candidata?
7. ¿Puede una herramienta de diseño de software identificar las dependencias funcionales? Explique brevemente su respuesta.
8. ¿Cuál es el significado de una FD con múltiples columnas que aparece en el lado derecho?
9. ¿Por qué se debe ser cauteloso cuando se usan FD con columnas múltiples en el lado izquierdo?

10. ¿Qué es una forma normal?
11. ¿Qué prohíbe la 1NF?
12. ¿Qué es una columna llave?
13. ¿Qué es una columna no llave?
14. ¿Qué tipos de FD no están permitidas en 2NF?
15. ¿Qué tipos de FD no están permitidas en 3NF?
16. ¿Cuál es la definición combinada de 2NF y 3NF?
17. ¿Qué tipos de FD no están permitidas en BCNF?
18. ¿Cuál es la relación entre BCNF y 3NF? ¿Es BCNF una forma más estricta que 3NF? Explique brevemente su respuesta.
19. ¿Por qué se prefiere la definición de BCNF a la de 3NF?
20. ¿Cuáles son los casos especiales que abarca BCNF pero no 3NF?
21. ¿Son significativos los casos especiales cubiertos por BCNF y no por 3NF?
22. ¿Cuál es el objetivo del procedimiento de síntesis simple?
23. ¿Qué es una limitación del procedimiento de síntesis simple?
24. ¿Qué es una dependencia transitiva?
25. ¿Se permiten las dependencias transitivas en las tablas 3NF? Explique por qué sí o por qué no.
26. ¿Por qué la eliminación de las dependencias transitivas de las FD se utiliza como información para el procedimiento de síntesis simple?
27. ¿Cuándo es necesario llevar a cabo el quinto paso del procedimiento de síntesis simple?
28. ¿De qué manera son semejantes el concepto de independencia de relación y el de independencia estadística?
29. ¿Qué tipo de redundancia se ocasiona por la independencia de relación?
30. ¿Cuántas columnas implica una MVD?
31. ¿Qué es una dependencia multivaluada (MVD)?
32. ¿Cuál es la relación entre MVD y FD?
33. ¿Qué es una MVD no trivial?
34. ¿Cuál es el objetivo de la 4NF?
35. ¿Cuáles son las ventajas del uso de la normalización como herramienta de refinamiento en lugar de una herramienta de diseño inicial?
36. ¿Por qué no se considera a la 5NF como una forma práctica?
37. ¿Por qué no se considera a DKNF como un formulario práctico?
38. ¿Cuándo es útil la desnormalización? Dé un ejemplo que muestre cuándo es beneficioso violar una 3NF.
39. ¿Cuáles son las dos formas de usar la desnormalización en el proceso de desarrollo de una base de datos?
40. ¿Por qué recomienda este libro el uso de la normalización como herramienta de refinamiento, no como herramienta de diseño inicial?

## Problemas

Además de los problemas que se presentan aquí, el estudio del caso que se presenta en el capítulo 13 proporciona práctica adicional. Para complementar los ejemplos de este capítulo, el capítulo 13 ofrece un caso completo de diseño de una base de datos, incluyendo el modelado de datos, la conversión de esquemas y la normalización.

**TABLA 7.P1 Datos muestra para la gran tabla de la base de datos de la universidad**

StdSSN	StdCity	StdClass	OfferNo	OffTerm	OffYear	EnrGrade	CourseNo	CrsDesc
S1	SEATTLE	JUN	O1	FALL	2006	3.5	C1	DB
S1	SEATTLE	JUN	O2	FALL	2006	3.3	C2	VB
S2	BOTHELL	JUN	O3	SPRING	2007	3.1	C3	OO
S2	BOTHELL	JUN	O2	FALL	2006	3.4	C2	VB

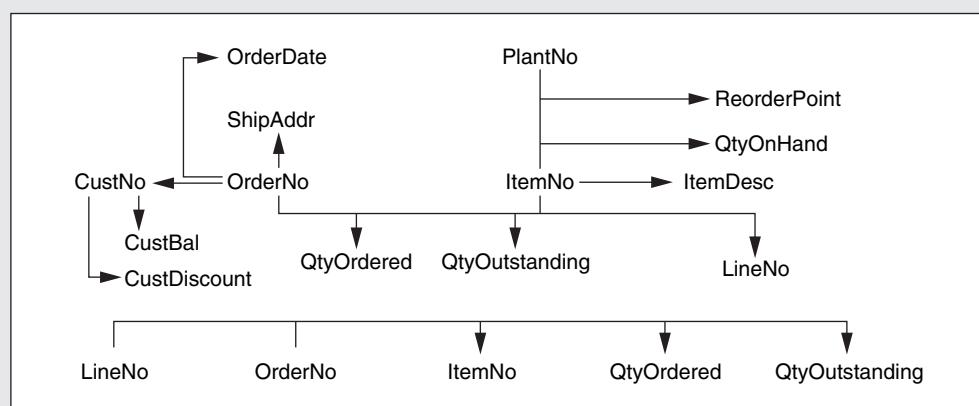
- Para la tabla de la gran base de datos de la universidad, enliste las FD con la columna *StdCity* como determinante de que *no* son verdaderas debido a los datos del ejemplo. Con cada FD que no se soporte, identifique las filas muestra que la contradigan. Recuerde que se necesitan dos registros para contradecir una FD. Los datos muestra se repiten en la tabla 7.P1 para su referencia.
- Siguiendo con el problema 1, enliste las FD con la columna *StdCity* como determinante de que no se violan los datos de ejemplo. Para cada FD añada uno o más registros de muestra y después identifique los datos muestra que contradigan la FD. Recuerde que se requieren dos registros para contradecir una FD.
- Para la gran tabla de pacientes, enliste las FD con la columna *PatZip* como la determinante de que *no* son verdaderas debido a los datos de ejemplo. Excluya la FD *PatZip* → *PatCity* porque es una FD válida. Con cada FD que no se soporte, identifique los registros de la muestra que la contradigan. Recuerde que son necesarios dos registros para contradecir una FD. Los datos de ejemplo se repiten en la tabla 7.P2 para su referencia.
- Siguendo con el problema 3, enliste las FDs con la columna *PatZip* como determinante de que los datos de ejemplo no violan. Excluya la FD *PatZip* → *PatCity* porque es una FD válida. Para cada FD añada uno o más registros de muestra y después identifique los datos muestra que contradigan la FD. Recuerde que se requieren dos registros para contradecir una FD.
- Aplique el procedimiento de síntesis simple a las FD de la gran tabla de pacientes. Las FD se repiten en la tabla 7.P3 para su referencia. Muestre el resultado de cada paso del procedimiento. Incluya en la lista de las tablas las llaves primarias, llaves foráneas y otras llaves candidatas.

**TABLA 7.P2** Datos de ejemplo para la gran tabla de pacientes

VisitNo	VisitDate	PatNo	PatAge	PatCity	PatZip	ProvNo	ProvSpecialty	Diagnosis
V10020	1/13/2007	P1	35	DENVER	80217	D1	INTERNIST	EAR INFECTION
V10020	1/13/2007	P1	35	DENVER	80217	D2	NURSE PRACTITIONER	INFLUENZA
V93030	1/20/2007	P3	17	ENGLEWOOD	80113	D2	NURSE PRACTITIONER	PREGNANCY
V82110	1/18/2007	P2	60	BOULDER	85932	D3	CARDIOLOGIST	MURMUR

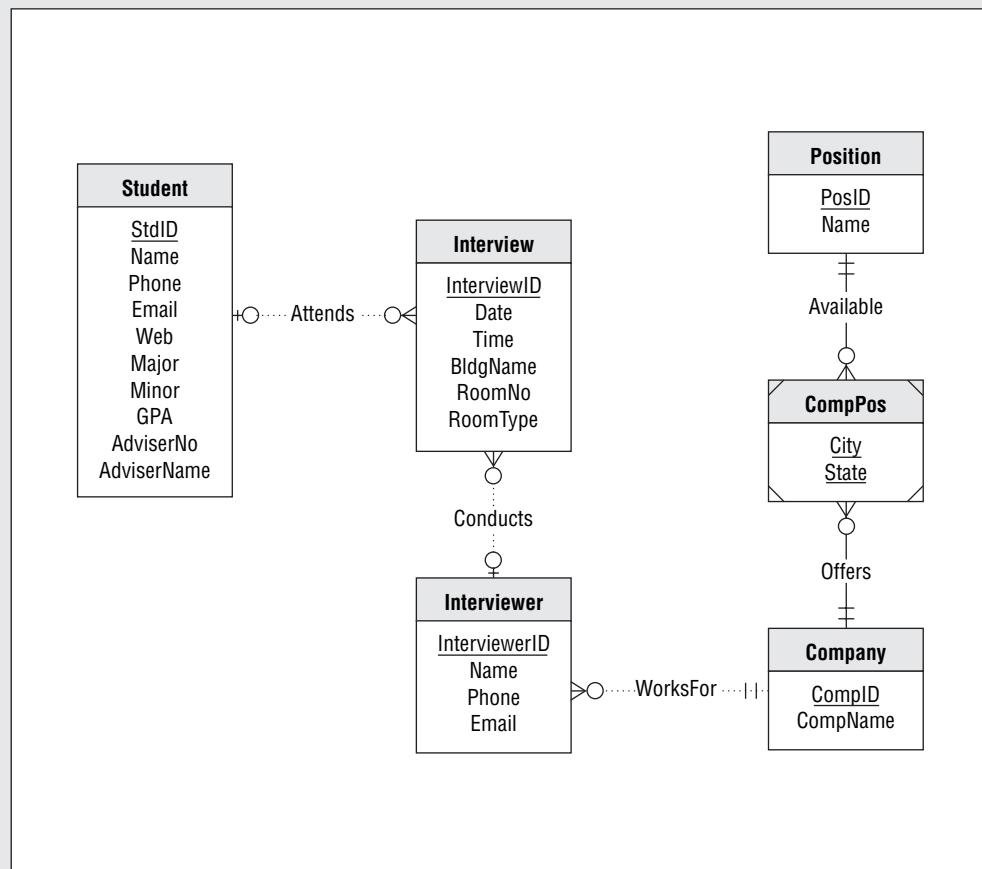
**TABLA 7.P2**  
Lista de FD para la  
gran tabla de pacientes

PatNo → PatAge, PatCity, PatZip  
 PatZip → PatCity  
 ProvNo → ProvSpecialty  
 VisitNo → PatNo, VisitDate, PatAge, PatCity, PatZip  
 VisitNo, ProvNo → Diagnosis

**FIGURA 7.P1**  
Diagrama de dependencia para la gran tabla de captura de órdenes

6. El diagrama de FD de la figura 7.P1 muestra FD entre las columnas de una bases de datos de captura de órdenes. La figura 7.P1 muestra FD con determinantes *CustNo*, *OrderNo*, *ItemNo*, la combinación de *OrderNo* e *ItemNo*, la de *ItemNo* y *PlantNo*, y la de *OrderNo* y *LineNo*. En las FD inferiores la combinación de *LineNo* y *OrderNo* determina *ItemNo* y la combinación de *OrderNo* e *ItemNo* determina *LineNo*. Para probar su comprensión de los diagramas de dependencia, convierta el diagrama de dependencia en una lista de dependencias organizada por los LHS.
7. Por medio del diagrama de FD (figura 7.P1) y la lista de FD (solución del problema 6) como lineamientos, haga una tabla con los datos de la muestra. Hay dos llaves candidatas para la tabla en cuestión: la combinación de *OrderNo*, *ItemNo* y *PlantNo*, y la combinación de *OrderNo*, *LineNo* y *PlantNo*. Usando datos de ejemplo, identifique las anomalías de inserción, actualización y eliminación de la tabla.
8. Derive tablas de 2NF a partir de la lista de FD del problema 6 y de la tabla del problema 7.
9. Derive tablas de 3NF a partir de la lista de FD del problema 6 y de las tablas de 2NF del problema 8.
10. Siguiendo con los problemas 6 y 7, aplique el procedimiento de síntesis simple para producir tablas BCNF.
11. Modifique su diseño de tablas del problema 10 si la columna de la dirección de envío (*ShipAddr*) determina el número de cliente (*CustNo*). ¿Piensa usted que esta FD adicional es razonable? Explique brevemente su respuesta.
12. Regrese al diagrama original de FD en donde *ShipAddr* no determina *CustNo*. ¿Cómo cambia el diseño de la tabla si usted quiere mantener el rastreo de una lista maestra de direcciones de envío para cada cliente? Considere que no quiere perder una dirección de envío cuando se elimina una orden.
13. Con el uso de la siguiente lista de FD para una base de datos simplificada de informes de gastos, identifique las anomalías de inserción, actualización y eliminación si todas las columnas estuvieran en la misma tabla (gran tabla de informe de gastos). Existen dos tablas candidatas para la gran tabla de informe de gastos: *ExpItemNo* (número de artículo de gasto) y la combinación de *CatNo* (número de categoría) y *ERNo* (número de informe de gasto). *ExpItemNo* es la llave primaria de la tabla.
  - *ERNo* → *UserNo*, *ERSubmitDate*, *ERStatusDate*
  - *ExpItemNo* → *ExpItemDesc*, *ExpItemDate*, *ExpItemAmt*, *CatNo*, *ERNo*
  - *UserNo* → *UserFirstName*, *UserLastName*, *UserPhone*, *UserEmail*
  - *CatNo* → *CatName*, *CatLimit*
  - *ERNo*, *CatNo* → *ExpItemNo*
  - *UserEmail* → *UserNo*
  - *CatName* → *CatNo*
14. Usando la lista de FD del problema 13, identifique las FD que violan la 2NF. Con el conocimiento de cuáles FD violan 2NF, diseñe una colección de tablas que satisfaga a 2NF pero no a 3NF.
15. Usando la lista de FD del problema 13, identifique las FD que violan la 3NF. Con el conocimiento de cuáles FD violan 2NF, diseñe una colección de tablas que satisfaga a 3NF.
16. Aplique el procedimiento de síntesis simple para producir tablas BCNF usando la lista de FD proporcionada en el problema 13. Muestre los resultados de cada paso del análisis.
17. Convierta el ERD de la figura 7.P2 en tablas y lleve a cabo mayor normalización conforme sea necesario. Después de convertir las tablas, especifique las FD para cada una de ellas. Ya que la llave primaria de cada tabla determina las otras columnas, sólo debe identificar las FD en las que LHS no es la llave primaria. Si una tabla no está en BCNF, explique por qué y divídala en dos o más tablas que sí lo estén.
18. Convierta el ERD de la figura 7.P3 en tablas y lleve a cabo la normalización conforme sea necesario. Después de la conversión, especifique las FD para cada tabla. Ya que la llave primaria de cada tabla determina las otras columnas, sólo debe identificar las FD en las que LHS no es la llave primaria. Si una tabla no está en BCNF, explique por qué y divídala en dos o más tablas que sí lo estén. Observe que en los tipos de entidad *Owner* y *Buyer*, la llave primaria (*SSN*) está incluida a pesar de que ha sido heredada del tipo de entidad *Person*.
19. Convierta el ERD de la figura 7.P4 en tablas y lleve a cabo la normalización conforme sea necesario. Después de la conversión especifique las FD para cada tabla. Ya que la llave primaria de cada tabla determina las otras columnas, sólo debe identificar las FD en las que LHS no es la llave primaria. Si una tabla no está en BCNF, explique por qué y divídala en dos o más tablas que sí lo estén. En el tipo

**FIGURA 7.P2**  
ERD para el problema 13



**FIGURA 7.P3**  
ERD para el problema 14

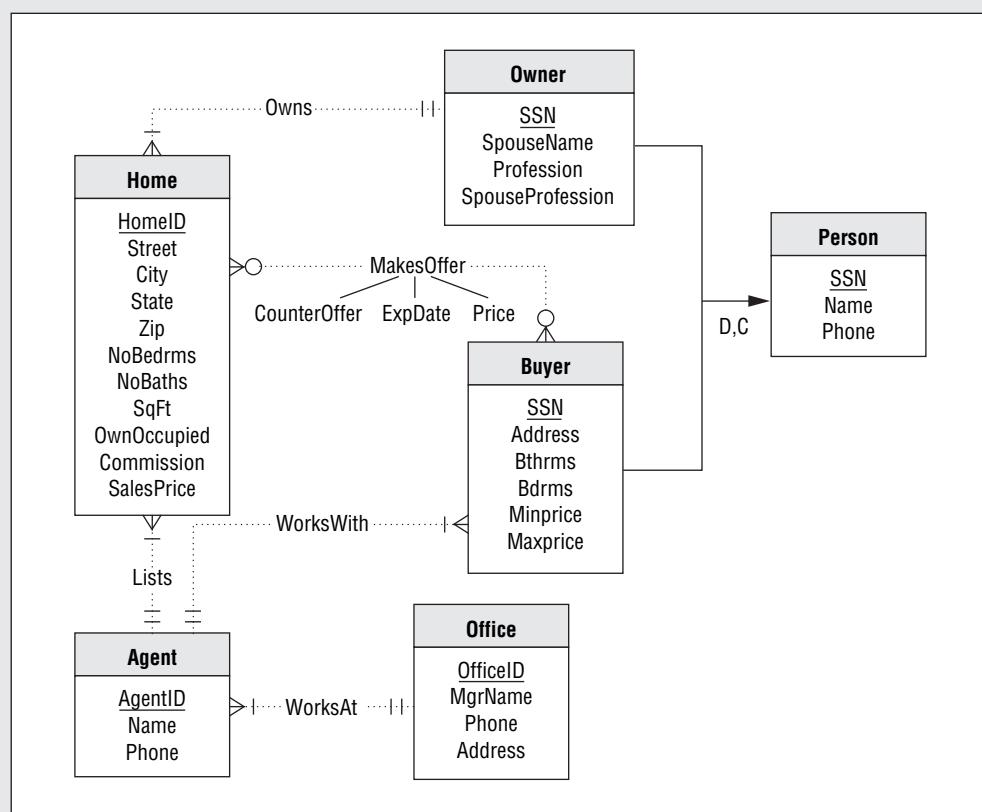


FIGURA 7.P4 ERD para el problema 15

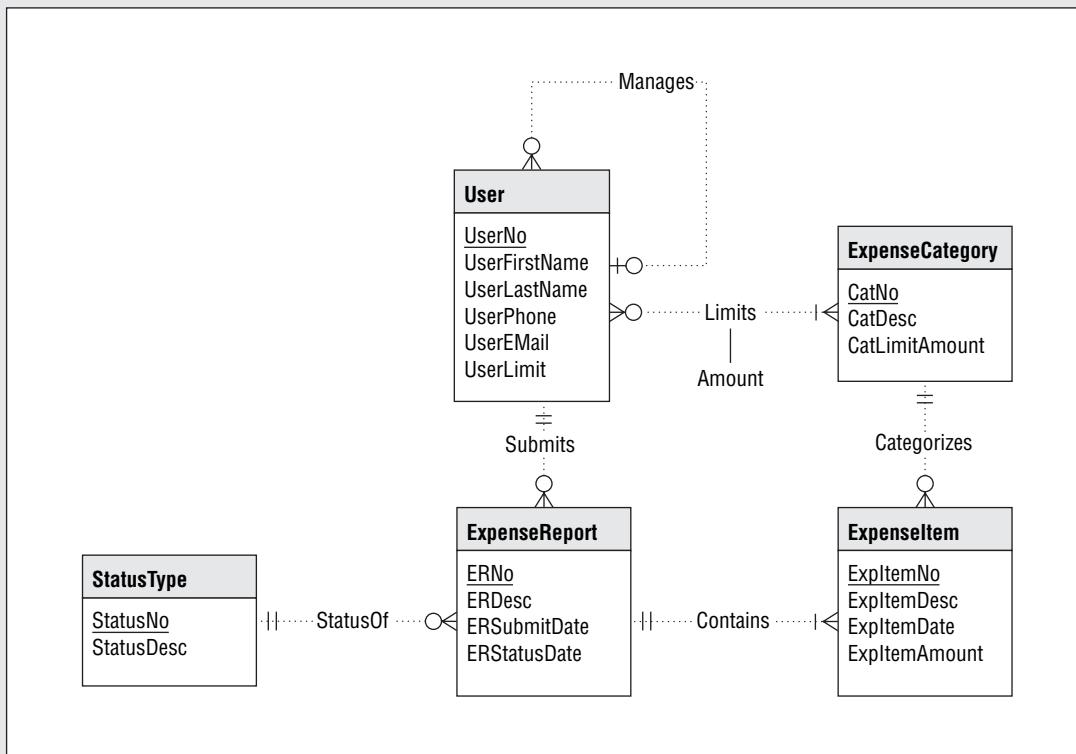
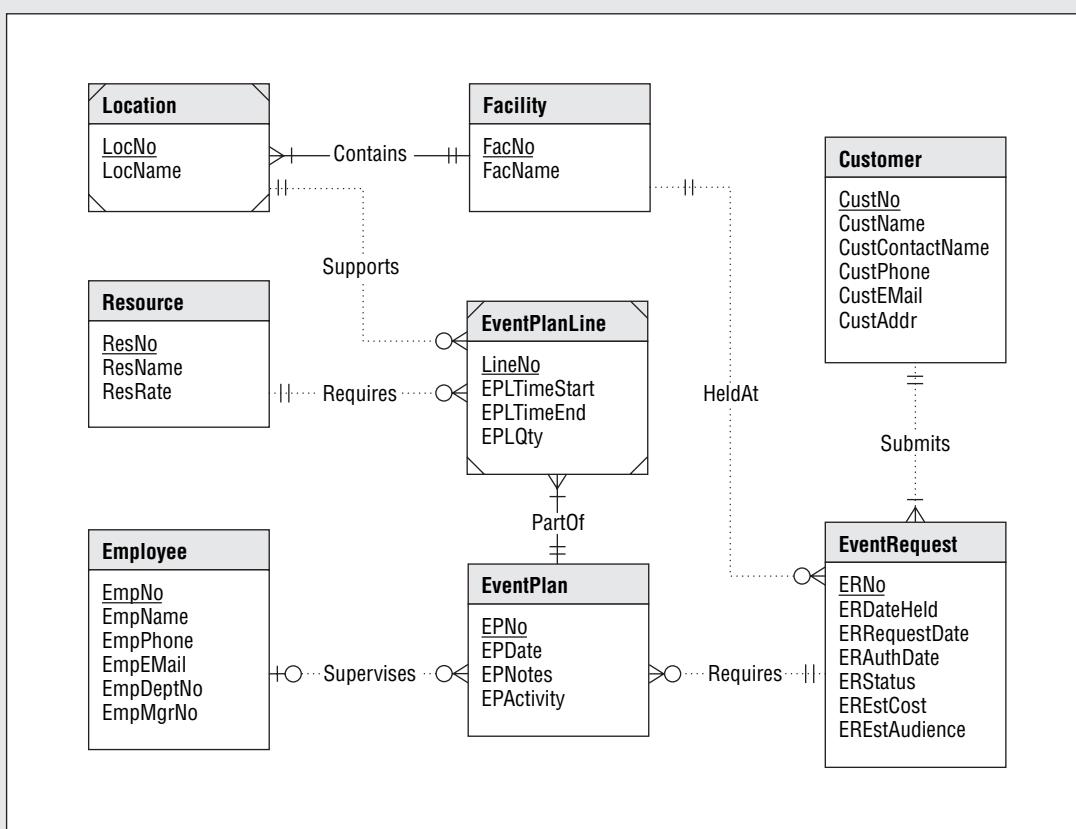


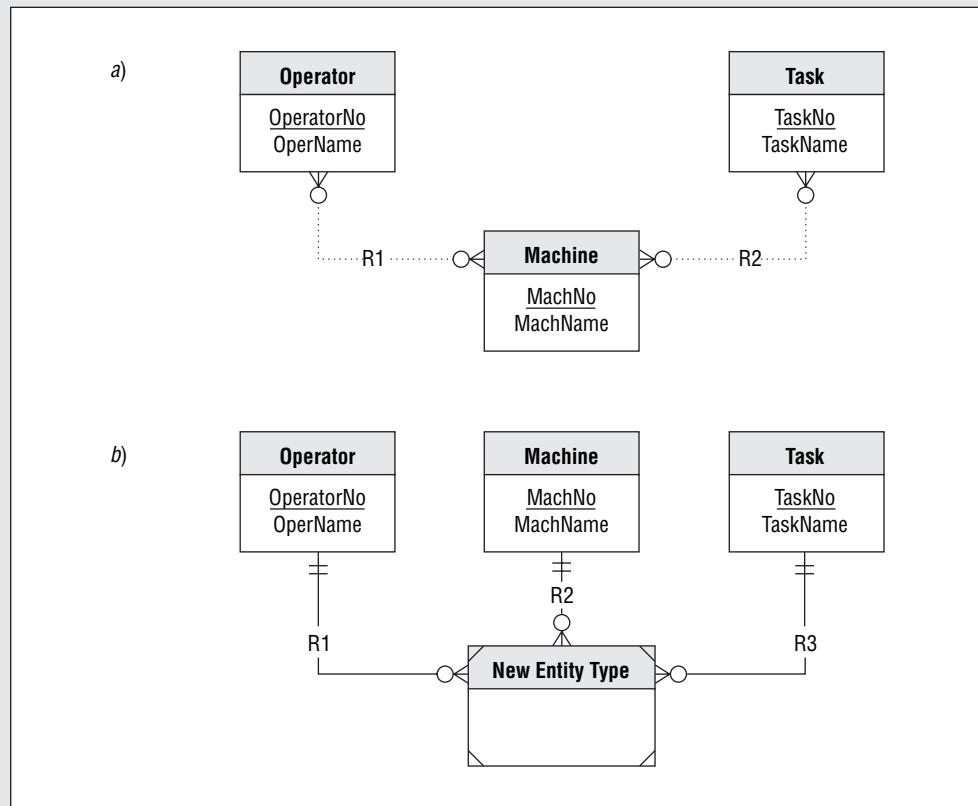
FIGURA 7.P5 ERD para el problema 16



- de entidad *User*, *UserEmail* es única. En el tipo de entidad *ExpenseCategory*, *CatDesc* es única. Para el tipo de entidad *ExpenseItem*, la combinación de las relaciones *Categorizes* y *Contains* es única.
20. Convierta el ERD de la figura 7.P5 en tablas y lleve a cabo la normalización conforme sea necesario. Después de la conversión especifique las FD para cada tabla. Ya que la llave primaria de cada tabla determina las otras columnas, sólo debe identificar las FD en las que LHS no es la llave primaria. Si una tabla no está en BCNF, explique por qué y divídala en dos o más tablas que sí lo estén. En el tipo de entidad *Employee*, cada departamento tiene un gerente. Todos los empleados de un departamento son supervisados por el mismo gerente. Para los otros tipos de entidad, *FacName* es única en *Facility*, *ResName* es única para *Resource* y *CustName* y *CustEmail* son únicas para *Customer*.
  21. Extienda la solución para el problema descrito en la sección 7.2.4 acerca de la base de datos que debe rastrear los artículos presentados en una conferencia. Las partes subrayadas en la descripción son nuevas. Escriba las nuevas FD. Por medio del procedimiento de síntesis simple diseñe una colección de tablas en BCNF. Observe las dependencias que no son importantes para el problema y suavice su diseño de BCNF conforme sea apropiado. Justifique su razonamiento:
    - La información del autor incluye un número único de autor, un nombre, una dirección de correo postal y una única pero opcional dirección de correo electrónico.
    - La información del artículo incluye la lista de autores, el autor principal, el número del documento, el título, el resumen, el estado de la revisión (pendiente, aceptado, rechazado) y una lista de categorías de temas.
    - La información del revisor incluye el número del revisor, el nombre, la dirección de correo postal, una única pero opcional dirección de correo electrónico, y un listado de categorías de conocimiento.
    - Una revisión completa incluye el número del revisor, la fecha, el número del artículo, los comentarios a los autores, comentarios al jefe del programa y clasificaciones (general, originalidad, corrección, estilo y relevancia).
    - Los artículos aceptados se asignan a sesiones. Cada sesión tiene un número identificador único, una lista de artículos, un orden de presentación para cada artículo, el título de la sesión, un presidente de sesión, la sala, la fecha, la hora de inicio y la duración. Observe que cada artículo aceptado puede ser asignado solamente a una sesión.
  22. Para la siguiente descripción de una base de datos de reservaciones de una línea aérea, identifique las dependencias funcionales y construya tablas normalizadas. Por medio del procedimiento de síntesis simple diseñe una colección de tablas en BCNF. Observe las dependencias que no son importantes para el problema y suavice su diseño de BCNF como sea adecuado. Justifique su razonamiento.
- Fly by Night Operation es una nueva aerolínea dirigida al creciente mercado de viajeros clandestinos (fugitivos, espías, artistas, sinvergüenzas, vagos, esposos engañadores, políticos, etc.). Fly by Night Operation necesita una base de datos para rastrear sus vuelos, clientes, tarifas, desempeño de la aeronave y asignación de personal. Como Fly by Night Operation es conocida como “una manera rápida de salir de la ciudad”, no se asignan asientos individuales y no se rastrean los vuelos de otras aerolíneas. A continuación se describen notas más específicas sobre las distintas partes de la base de datos:
- La información sobre el vuelo incluye un número único de vuelo, su origen, su (supuesto) destino, y los horarios de salida y llegada estimados (vagamente). Para reducir los costos, Fly by Night Operation solamente cuenta con vuelos sin escalas con un origen y destino únicos.
  - Los vuelos se programan para una o más fechas con una aeronave y tripulación asignados a cada vuelo programado y el resto de la capacidad del avión estipulada (los asientos que sobran). En una asignación de tripulación, se considera el número de empleado y su función (capitán, aeromoza).
  - Los aviones tienen un único número de serie, un modelo, una capacidad, y la fecha programada para su próximo mantenimiento.
  - El registro de mantenimiento de un avión incluye un único número de mantenimiento, una fecha, una descripción, el número de serie del avión y el empleado responsable de la reparación.
  - Los empleados tienen un único número de empleado, un nombre, un teléfono y un puesto.
  - Los clientes tienen un número de cliente, un teléfono y un nombre (generalmente un alias).
  - Se conservan registros para las reservaciones de vuelos programados incluyendo un único número de reserva, el número de vuelo, la fecha de vuelo, el número del cliente, la fecha de reserva, la tarifa y el método de pago (por lo general efectivo, pero ocasionalmente el cheque de alguien más o tarjeta de crédito). Si el pago se hace con tarjeta de crédito, el número de tarjeta y la fecha de expiración de la misma, forman parte del registro de la reserva.

23. Para la siguiente descripción de una base de datos para llevar la contabilidad, identifique las dependencias funcionales y construya tablas normalizadas. Por medio del procedimiento de síntesis simple diseñe una colección de tablas en BCNF. Observe las dependencias que no son importantes para el problema y suavice su diseño a partir de BCNF conforme sea adecuado. Justifique su razonamiento.
- La función primaria de la base de datos es grabar las entradas a un registro. Un usuario puede tener varias cuentas y hay un registro para cada una de ellas.
  - La información acerca de los usuarios incluye un único número de usuario, un nombre, la calle de su domicilio, una ciudad, un estado, un código postal y una única pero opcional dirección de correo electrónico.
  - Las cuentas tienen atributos que implican un único número de cuenta, un nombre único, una fecha de inicio, el número del último cheque, el tipo de cuenta (de cheques, de inversiones, etc.), un número de usuario y un saldo actual (calculado). Para las cuentas de cheques, también se registran el número del banco (único), el nombre del mismo y su dirección.
  - Una entrada incluye un único número, un tipo, un número de cheque opcional, un pagador, una fecha, una cantidad, una descripción, un número de cuenta y una lista de las líneas de entrada. El tipo puede tener diversos valores incluyendo ATM, número del siguiente cheque, depósito, tarjeta de débito.
  - En la lista de líneas de entrada, el usuario coloca la cantidad total de la entrada a las categorías. Una línea de entrada incluye el nombre de la categoría, una descripción de la línea de entrada y una cantidad.
  - Las categorías tienen otros atributos que no se muestran en una línea de entrada: un número único de categoría (el nombre también es único), una descripción, un tipo (activo, gasto, utilidad o pasivo), y un estado fiscal (sí o no).
  - Las categorías están organizadas en jerarquías. Por ejemplo, hay una categoría Auto con subcategorías Auto:fuel y Auto:repair. Las categorías tienen diversos niveles de subcategorías.
24. Para los ERD de la figura 7.P6, describa las consideraciones bajo las cuales los ERD muestran de manera correcta las relaciones entre operadores, máquinas y tareas. En cada caso, seleccione los

**FIGURA 7.P6**  
ERD para el problema 24



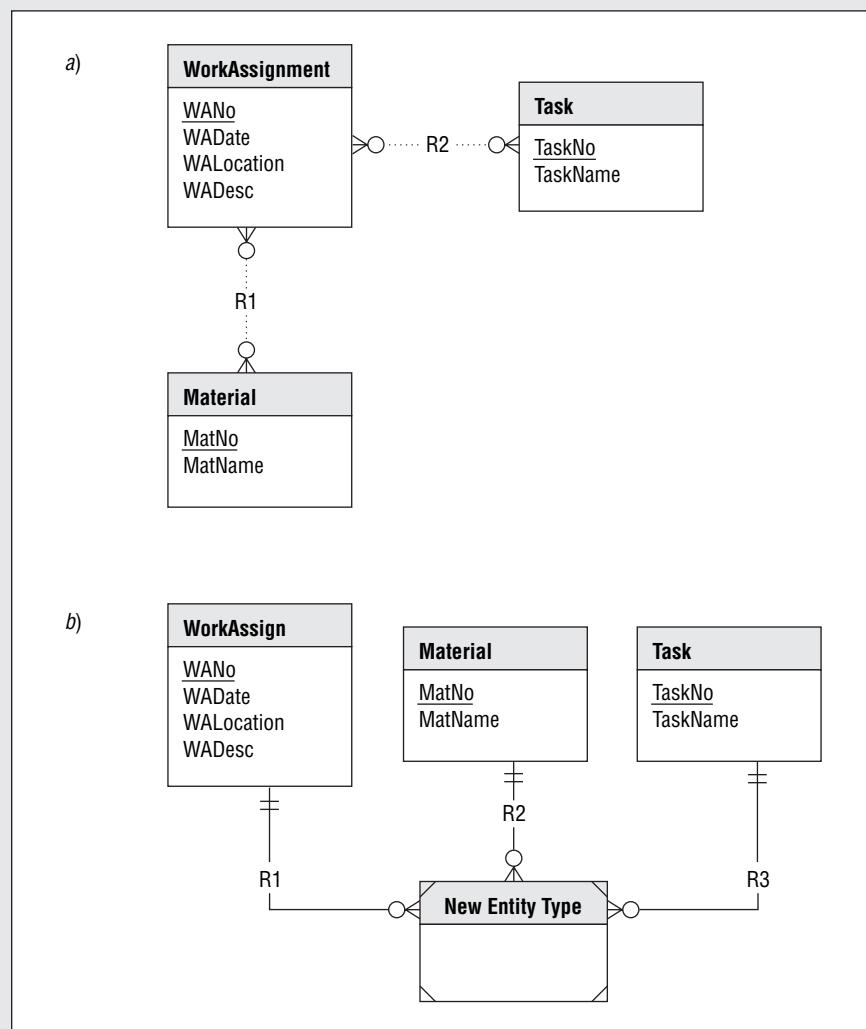
nombres adecuados para las relaciones y describa el significado de las mismas. En la parte (b) deberá elegir el nombre para un nuevo tipo de entidad.

25. Para la siguiente descripción de una base de datos diseñada para dar apoyo a las operaciones físicas de la planta, identifique las dependencias funcionales y construya tablas normalizadas. Por medio del procedimiento de síntesis simple, diseñe una colección de tablas en BCNF. Observe las dependencias que no son importantes para el problema y suavice su diseño a partir de BCNF conforme sea adecuado. Justifique su razonamiento.

Diseñe una base de datos que ayude al personal físico de la planta a administrar las tarjetas llave para acceder a los edificios y las salas. El propósito principal de la base de datos es asegurar el conteo preciso de todas las tarjetas llave.

- Un edificio tiene un número único de edificio, nombre y ubicación dentro del campus.
- Una sala tiene un único número, tamaño (dimensiones físicas), capacidad, número de entradas, y una descripción del equipo que se encuentra dentro de la sala. Cada sala está localizada exactamente en un edificio. El número de sala incluye la identificación del edificio en el que se encuentra y se sigue por un número entero. Por ejemplo, el número de sala KC100 identifica a la sala 100 en el edificio King Center (KC).
- Un empleado tiene un único número de empleado, un nombre, un puesto, una dirección de correo electrónico, un teléfono y un número opcional de sala en la que trabaja.
- Las tarjetas de acceso codificadas magnéticamente están diseñadas para abrir una o varias salas. Una tarjeta de acceso posee un único número de tarjeta, una fecha codificada, una lista de números de salas que puede abrir y el número del empleado que está autorizado para su uso. Una sala podría tener una o más tarjetas que la abran. Un tipo de llave debe autorizarse antes de ser creada.

**FIGURA 7.P7**  
ERD para el problema 26



26. Para las ERD de la figura 7.P7, describa consideraciones bajo las cuales las ERD muestran correctamente las relaciones entre las asignaciones de trabajo, tareas y materiales. Una asignación de trabajo contiene el trabajo asignado para una construcción en un lugar específico. El trabajo programado incluye las tareas y los materiales necesarios para la construcción. En cada caso, elija nombres apropiados para las relaciones y describa el significado de las mismas. En la parte (b) tendrá que elegir el nombre para un nuevo tipo de entidad.
27. Para la siguiente descripción de una base de datos diseñada para dar apoyo al rastreo de voluntarios, identifique las dependencias funcionales y construya tablas normalizadas. Por medio del procedimiento de síntesis simple, diseñe una colección de tablas en BCNF. Observe las dependencias que no son importantes para el problema y suavice su diseño a partir de BCNF conforme sea adecuado. Justifique su razonamiento.

Diseñe una base de datos que apoye a las organizaciones que necesitan encontrar voluntarios, áreas de voluntarios, eventos y horas trabajadas en los eventos. Inicialmente el sistema se empleará para escuelas que obligan a la participación de los padres de los alumnos como voluntarios. Los padres se registran como una familia de uno o dos padres. Los coordinadores de voluntarios los reclutan de las áreas de voluntarios. Los organizadores de eventos reclutan a los voluntarios para que trabajen en los eventos. Algunos eventos requieren de un horario de voluntarios mientras que otros no lo utilizan. Los voluntarios trabajan en los eventos y registran el tiempo que han trabajado.

- Para cada familia la base de datos registra el número único de familia, el nombre y apellido de cada uno de los padres, los teléfonos de casa y oficina, la dirección (calle, ciudad, estado y código postal) y una dirección opcional de correo electrónico. En el caso de las familias con un solo parente, se registra solamente la información de un parente.
- Para cada área de voluntarios, la base de datos registra el único número de área de voluntarios, el nombre de la misma, el grupo (claustro de profesores o asociación de padres de familia) que controla el área de voluntarios y la familia que la coordina. En algunos casos, una familia podría coordinar más un área de voluntarios.
- Para los eventos, la base de datos registra un único número de evento, la descripción del mismo, la fecha en que se lleva a cabo, la hora de inicio y término del evento, el número de voluntarios requeridos, la periodicidad y fecha de expiración si se trata de un evento recurrente, el área de voluntarios y la lista de familias voluntarias para el evento. Las familias pueden programar su participación en un grupo de eventos.
- Las horas trabajadas se registran después de completar un trabajo asignado. La base de datos contiene el nombre y apellido del voluntario, la familia a la que representa, el número de horas trabajadas, el evento opcional, la fecha de trabajo, la ubicación del trabajo y comentarios opcionales. Por lo regular, el voluntario es uno de los padres de la familia, pero ocasionalmente el voluntario puede ser amigo o parente de la familia. El evento es opcional para permitir que haya horas de trabajo voluntario que no sean consideradas como eventos.

## Referencias para ampliar su estudio

El tema de la normalización puede detallarse mucho más de lo que se ha descrito en este capítulo. Para obtener una descripción más detallada de este tema, consulte libros de ciencias computacionales, como Elmasri y Navathe (2004). El tema de procedimiento de síntesis simple fue adaptado de Hawryszkiewycz (1984). Para una tutoría clásica sobre normalización, consulte Kent (1983). Fagin (1981) describe la forma normal de la llave de dominio, la forma normal definitiva. Los sitios *DBAZine* ([www.dbazine.com/](http://www.dbazine.com/)) y la *DevX Database Zone* ([www.devx.com](http://www.devx.com)) ofrecen consejos prácticos sobre el desarrollo y la normalización de una base de datos.

# Capítulo

# 8

# Diseño físico de bases de datos

## Objetivos de aprendizaje

Este capítulo describe el diseño físico de bases de datos, la fase final del proceso de desarrollo de bases de datos. El diseño físico de bases de datos transforma el diseño de tablas de la fase del diseño lógico en una implementación eficiente que soporte todas las aplicaciones que usen la base de datos. Al finalizar este capítulo, el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Describir las entradas, salidas y objetivos del diseño físico de bases de datos.
- Apreciar las dificultades de realizar un diseño físico de bases de datos y la necesidad de revisiones periódicas a las opciones del diseño físico de bases de datos.
- Enlistar las siguientes características de las estructuras de archivos: secuencial, Btree, hash y bitmap.
- Comprender las alternativas que sigue el optimizador de consultas y las áreas en las que se pueden mejorar las decisiones de optimización.
- Comprender las ventajas-desventajas de las decisiones de selección de índices y desnormalización.
- Comprender la necesidad del uso de herramientas asistidas por computadora para ayudarle con las decisiones del diseño físico de bases de datos, en especial, con las decisiones afectadas por el proceso de optimización de consultas.

## Panorama general

Los capítulos 5 a 7 cubrieron las fases conceptual y del diseño lógico del desarrollo de bases de datos. Usted aprendió sobre los diagramas de entidad-relación, prácticas del modelado de datos, conversión de esquemas y normalización. Este capítulo amplía sus habilidades sobre el diseño de bases de datos explicando el proceso para realizar la implementación eficiente del diseño de tablas.

Para ser más eficiente en el diseño físico de bases de datos, necesita comprender el proceso y el entorno. Este capítulo describe el proceso del diseño físico de bases de datos incluyendo entradas, salidas y objetivos, junto con dos conceptos críticos del entorno: estructuras de archivos y optimización de consultas. La mayoría de las opciones del diseño físico de bases de datos relacionan las características de las estructuras de archivos y las decisiones sobre la optimización de consultas.

Después de entender el proceso y el entorno, estará listo para llevar a cabo un diseño físico de bases de datos. Al hacer un diseño físico de bases de datos, deberá proporcionar las entradas detalladas y escoger opciones para balancear las necesidades de recuperación y actualización de las aplicaciones. Este capítulo describe la complejidad de los perfiles de las tablas y de los perfiles de las aplicaciones, así como su importancia para las decisiones del diseño físico de bases de datos. La selección de índices es la alternativa más importante del diseño físico de bases de datos. Este capítulo describe las ventajas y desventajas de la selección de índices y proporciona las reglas de selección de índices que puede aplicar a bases de datos de tamaño moderado. Además de la selección de índices, este capítulo presenta la desnormalización, formateo de registros y procesamiento en paralelo como técnicas para mejorar el desempeño de las bases de datos.

## 8.1 Panorama general del diseño físico de bases de datos

Las decisiones en la fase del diseño físico de bases de datos involucran el nivel de almacenamiento de una base de datos. De forma colectiva, a las decisiones sobre el nivel de almacenamiento se les conoce como el esquema interno. Esta sección describe el nivel de almacenamiento y los objetivos, entradas y salidas del diseño físico de una base de datos.

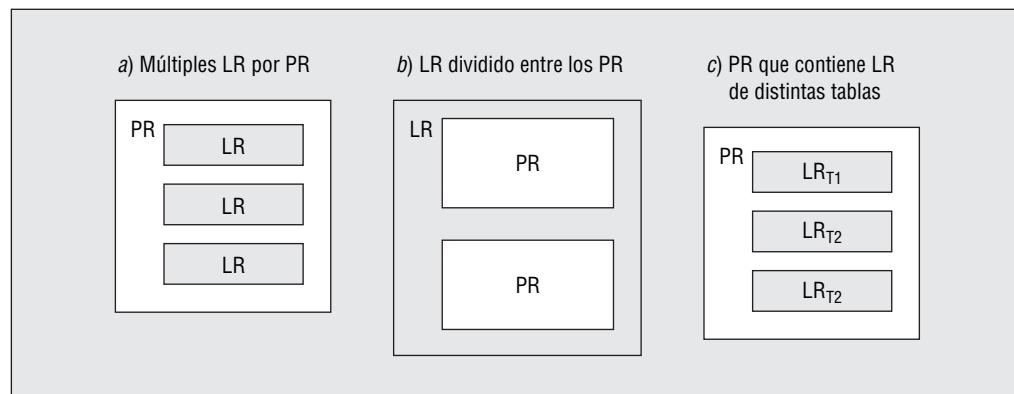
### 8.1.1 Niveles de almacenamiento de las bases de datos

**registro físico**  
conjunto de bytes que se transfieren entre el almacenamiento volátil de la memoria principal y el almacenamiento fijo de un disco. El número de accesos a los registros físicos es una medida importante del desempeño de la base de datos.

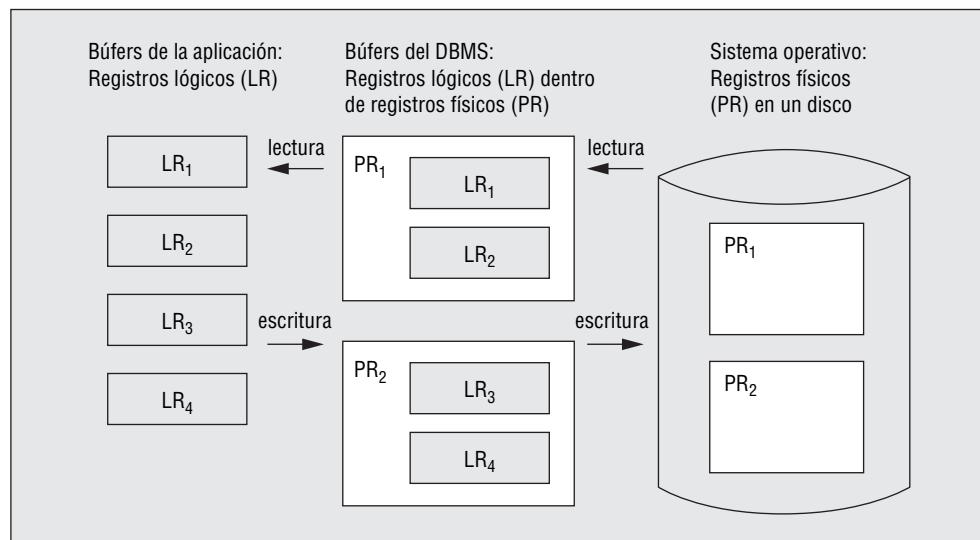
El nivel de almacenamiento está más cerca del hardware y del sistema operativo. En el nivel de almacenamiento, una base de datos está formada de registros físicos (también conocidos como bloques o páginas) organizados en archivos. Un registro físico es un conjunto de bytes que se transfieren entre el almacenamiento volátil de la memoria principal y el almacenamiento fijo de un disco. A la memoria principal se le considera como almacenamiento volátil porque los contenidos de la memoria principal se pueden perder si ocurre alguna falla. Un archivo es un conjunto de registros físicos organizados para conseguir un acceso eficiente. La figura 8.1 ilustra las relaciones entre los registros lógicos (filas de una tabla) y registros físicos almacenados en un archivo. Generalmente, un registro físico contiene varios registros lógicos. El tamaño de un registro físico es una potencia del número dos, tal como 1 024 ( $2^{10}$ ) o 4 096 ( $2^{12}$ ) bytes. Un registro lógico más grande se puede dividir entre varios registros físicos. Otra posibilidad es que los registros lógicos de más de una tabla se almacenen en el mismo registro físico.

El DBMS y el sistema operativo trabajan de manera conjunta para satisfacer las solicitudes de registros lógicos hechas por las aplicaciones. La figura 8.2 ilustra el proceso de transferencia de registros físicos y lógicos entre un disco, búfers del DBMS y búfers de la aplicación. Normalmente el DBMS y la aplicación tienen áreas de memoria separadas conocidas como búfers. Cuando una aplicación hace una solicitud para un registro lógico, el DBMS ubica al registro físico que lo contiene. En el caso de una operación de lectura, el sistema operativo transfiere el registro físico del disco al área de memoria del DBMS. Después el DBMS transfiere el registro lógico al búfer de la aplicación. El proceso de transferencia se invierte en el caso de una operación de escritura.

**FIGURA 8.1**  
**Relaciones entre los registros lógicos (LR) y los registros físicos (PR)**



**FIGURA 8.2**  
Transferencia de registros físicos



Un requerimiento de registro lógico puede no resultar en una transferencia de registro físico debido al proceso denominado *buffering*. El DBMS intenta anticiparse a las necesidades de las aplicaciones para que los registros físicos correspondientes residan ya en los búferes del DBMS. Una dificultad significativa acerca de la predicción del desempeño de la base de datos es conocer cuándo una solicitud de registro lógico conduce a una transferencia de registro físico. Por ejemplo, si varias aplicaciones están accediendo a los mismos registros lógicos, los registros físicos correspondientes pueden residir en los búferes del DBMS. Consecuentemente, la incertidumbre acerca de los contenidos de los búferes del DBMS puede dificultar el diseño físico de la base de datos.

### 8.1.2 Objetivos y restricciones

El objetivo del diseño físico de bases de datos es minimizar los tiempos de respuesta para acceder y modificar una base de datos. Debido a que el tiempo de respuesta es difícil de estimar de modo directo, la minimización de los recursos de cómputo se utiliza como medida sustituta. Los recursos que consume el procesamiento de la base de datos son la transferencia de registros físicos, las operaciones de la unidad central de procesamiento (CPU), la memoria principal y el espacio en disco. Los últimos dos recursos (memoria principal y espacio en disco) son considerados como restricciones, en lugar de recursos a minimizar. La minimización de la memoria principal y del espacio en disco puede ocasionar tiempos largos de respuesta.

El número de accesos a los registros físicos limita el desempeño de la mayoría de las aplicaciones de bases de datos. Un acceso a un registro físico puede involucrar un movimiento mecánico del disco incluyendo la rotación y el movimiento de las cabezas magnéticas. El movimiento mecánico generalmente es mucho más lento que el intercambio electrónico de la memoria principal. La velocidad del acceso a disco se mide en milisegundos (milésimas de un segundo), mientras que el acceso a la memoria se mide en nanosegundos (millonésimas de un segundo). Por lo tanto, el acceso a un registro físico puede ser muchas veces más lento que el acceso a la memoria principal. La reducción del número de accesos a registros físicos generalmente mejorará el tiempo de respuesta.

El uso de CPU también puede ser un factor en algunas aplicaciones de bases de datos. Por ejemplo, la ordenación requiere un gran número de comparaciones y asignaciones. Estas operaciones, realizadas por el CPU, son mucho más rápidas que los accesos a registros físicos. Para balancear tanto los accesos a registros físicos como el uso del CPU, se puede usar un peso para combinarlos en una medida. El peso generalmente es algo cercano al 0 para reflejar que muchas operaciones del CPU se pueden realizar en el tiempo en el que se realiza una transferencia de registro físico.

**medida combinada del desempeño de una base de datos**  
 $PRA + W * CPU-OP$  en donde  $PRA$  es el número de accesos a registros físicos,  $CPU-OP$  es el número de operaciones del CPU, tales como comparaciones y asignaciones, y  $W$  es el peso, un número real entre 0 y 1.

El objetivo del diseño físico de bases de datos es minimizar la medida combinada para todas las aplicaciones que usan la base de datos. Por lo general, las mejoras en el desempeño de las aplicaciones de recuperación ocasionan un gasto en las aplicaciones de actualización y viceversa. Por lo mismo, un tema importante del diseño físico de bases de datos es balancear las necesidades de recuperación y actualización de las aplicaciones.

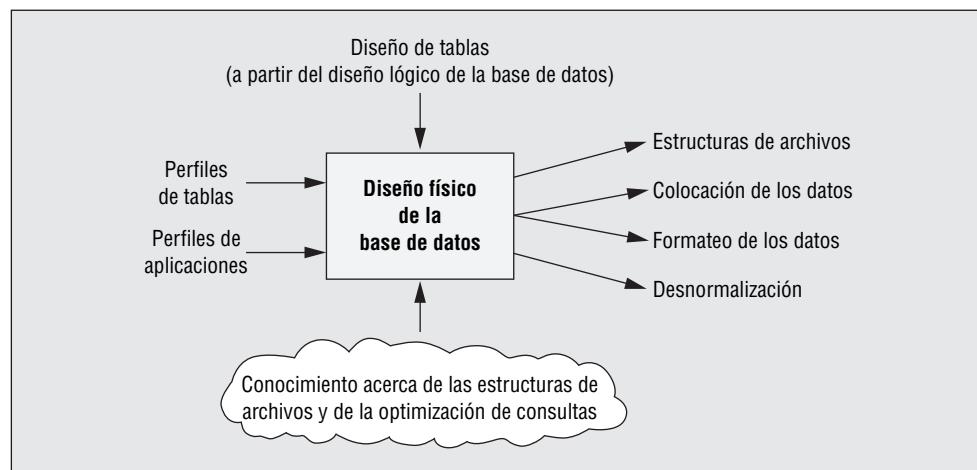
Las medidas de desempeño son demasiado detalladas como para estimarlas a mano, a excepción de algunas situaciones. El software de optimización compleja generalmente es parte del compilador de SQL. La comprensión de las medidas de desempeño le ayudará a interpretar las opciones presentadas por el software de optimización.

Para la mayoría de las opciones del diseño físico de bases de datos, las cantidades de memoria principal y de espacio en disco usualmente son fijas. En otras palabras, la memoria principal y el espacio en disco son restricciones del proceso de diseño físico de bases de datos. Como con las restricciones de otros problemas de optimización, usted debe considerar los efectos ocasionados cuando se modifican las cantidades proporcionadas de memoria principal y espacio en disco. Incrementar las cantidades de estos recursos puede mejorar el desempeño. La cantidad de mejora del desempeño puede depender de muchos factores tales como el DBMS, el diseño de las tablas y las aplicaciones que usa la base de datos.

### 8.1.3 Entradas, salidas y entorno

El diseño físico de bases de datos está formado por varias entradas y salidas distintas tal como se ilustra en la figura 8.3 y se resume en la tabla 8.1. El punto de partida es el diseño de tablas

**FIGURA 8.3**  
Entradas, salidas y entorno del diseño físico de bases de datos



**TABLA 8.1**  
Resumen de entradas, salidas y entorno del diseño físico de bases de datos

Elemento	Descripción
<b>Entradas</b>	
Perfil de tablas	Estadísticas para cada tabla, como número de filas y de columnas de valores únicos
Perfil de aplicación	Estadísticas para cada formulario, reporte y consulta, tales como accesos/actualizaciones y la frecuencia de los accesos/actualizaciones
<b>Salidas</b>	
Estructuras de archivos	Método de organización de registros físicos para cada tabla
Colocación de datos	Criterios para acomodar los registros físicos de forma cercana
Formateo de datos	Uso de la compresión y de los datos derivados
Desnormalización	Combinación de tablas separadas en una sola tabla
<b>Conocimiento del entorno</b>	
Estructuras de archivos	Características como las operaciones respaldadas y fórmulas de costo
Optimización de consultas	Decisiones de accesos hechas por el componente de optimización para cada una de las consultas

a partir de la fase del diseño lógico. Los perfiles de las tablas y aplicaciones se usan específicamente para el diseño físico de bases de datos. Debido a que estas entradas son críticas para el proceso de diseño físico de bases de datos, se comentan con mayor detalle en la sección 8.2. Las salidas más importantes son las decisiones de las estructuras de archivos y la colocación de los datos. La sección 8.5 describe estas decisiones con mayor detalle. Por simplicidad, las decisiones acerca de otras salidas se hacen de forma separada, incluso aunque las salidas estén relacionadas. Por ejemplo, las estructuras de archivos generalmente se seleccionan de forma separada de las decisiones de desnormalización, incluso aunque las decisiones de desnormalización puedan afectar las decisiones sobre las estructuras de archivos. Por ende, el diseño físico de bases de datos se representa mejor como una secuencia de procesos de toma de decisiones en lugar de un solo proceso enorme.

El conocimiento acerca de las estructuras de archivos y la optimización de consultas están dentro del entorno del diseño físico de bases de datos en lugar de ser entradas del mismo. El conocimiento se puede incluir dentro de las herramientas de diseño de bases de datos. Si no están disponibles las herramientas de diseño de bases de datos, el diseñador de manera informal utiliza el conocimiento que tenga del ambiente para tomar decisiones acerca de la base de datos física. La adquisición de conocimiento puede ser difícil, ya que mucho de ello es específico a cada DBMS. Debido a que el conocimiento del entorno es tan crucial en el proceso de diseño físico de bases de datos, las secciones 8.3 y 8.4 lo describen con mayor detalle.

### 8.1.4 Dificultades

Antes de proceder al conocimiento detallado del diseño físico de bases de datos, es importante comprender por qué es difícil. La dificultad se debe al número de decisiones, relaciones entre las decisiones, entradas detalladas, complejidad del entorno e incertidumbre para predecir los accesos a los registros físicos. Estos factores se discuten brevemente más adelante. En el resto de este capítulo, mantenga estas dificultades en mente.

- El número de posibles opciones a disposición del diseñador puede ser grande. Para bases de datos que tengan muchas columnas, el número de posibles opciones puede ser demasiado grande como para evaluarlo, incluso con computadoras grandes.
- Las decisiones no se pueden hacer de forma aislada entre ellas. Por ejemplo, las decisiones sobre la estructura de archivos de una tabla pueden influir en las decisiones de otras tablas.
- La calidad de las decisiones se limita a la precisión de los perfiles de las tablas y aplicaciones. Sin embargo, estas entradas pueden ser difíciles de obtener. Además, las entradas se modifican con el tiempo, por lo que es necesaria su obtención periódica.
- El conocimiento del entorno es específico para cada DBMS. Gran parte del conocimiento es un secreto de la marca registrada o demasiado complejo como para entender el detalle.
- El número de accesos a los registros físicos es difícil de predecir dada la incertidumbre de los contenidos de los búferes del DBMS. La incertidumbre surge debido a que la combinación de aplicaciones que acceden a la base de datos cambia constantemente.

## 8.2 Entradas del diseño físico de bases de datos

---

El diseño físico de bases de datos requiere de entradas específicas con detalles suficientes. Las entradas no especificadas con el detalle suficiente pueden llevar a tomar pobres decisiones en el diseño físico de bases de datos y la optimización de consultas. Esta sección describe el nivel de detalle recomendado tanto para los perfiles de las tablas como los perfiles de las aplicaciones.

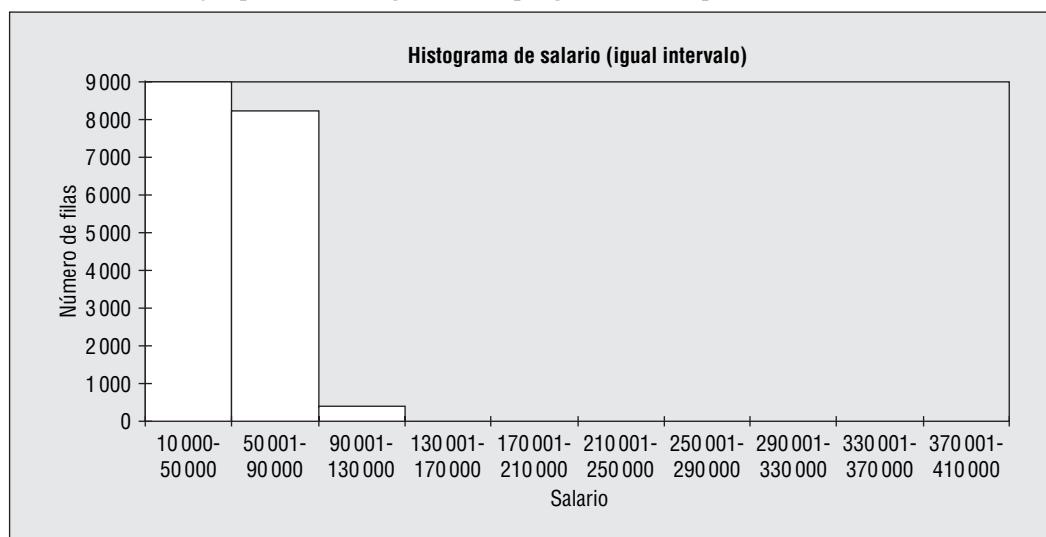
### 8.2.1 Perfiles de las tablas

El perfil de una tabla resume una tabla como un todo, las columnas dentro de la tabla y la relación entre las tablas, como se muestra en la tabla 8.2. Debido a que los perfiles de las tablas son tediosos como para construirlos de forma manual, la mayoría de los DBMS proporcionan

**TABLA 8.2**  
**Componentes típicos**  
**del perfil de una tabla**

Componente	Estadísticas
Tabla	Número de filas y registros físicos
Columna	Número de valores únicos, distribución de valores, correlación entre columnas
Relación	Distribución del número de filas relacionadas

**FIGURA 8.4** Ejemplo de un histograma del tipo igual-intervalo para la columna salario

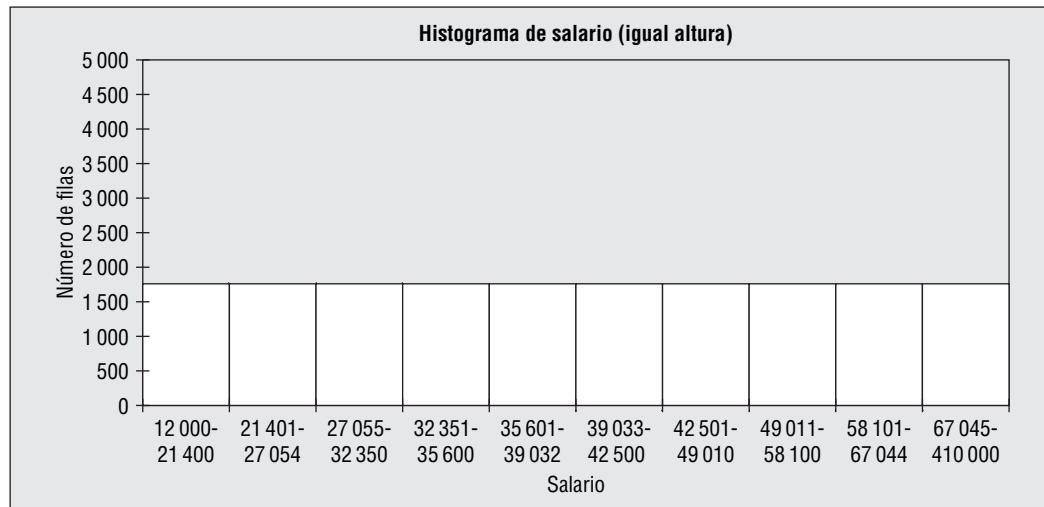


programas estadísticos para construirlos de forma automática. El diseñador puede necesitar de forma periódica ejecutar los programas de estadísticas para que los perfiles no se vuelvan obsoletos. Para las bases de datos grandes, los perfiles de las tablas se pueden estimar con ejemplos de la base de datos. Utilizar la base de datos completa puede consumir demasiado tiempo.

Para los resúmenes de columnas y relaciones, la distribución expresa el número de filas y filas relacionadas para los valores de las columnas. La distribución de valores se puede especificar de varias maneras. Una forma sencilla es asumir que los valores de las columnas están distribuidos de modo uniforme. La distribución uniforme significa que cada valor tiene un número igual de filas. Si se asume el valor uniforme, sólo se necesitan los valores mínimo y máximo.

Una forma más detallada para especificar una distribución es usar un histograma. Un histograma es una gráfica de dos dimensiones en la que el eje x representa los rangos de las columnas y el eje y representa el número de filas. Por ejemplo, la primera barra de la figura 8.4 indica que 9 000 filas tienen un salario entre 10 000 y 50 000 dólares. Los histogramas tradicionales del mismo intervalo no funcionan bien con datos asimétricos, ya que es necesario un gran número de rangos para controlar los errores estimados. En la figura 8.4, la estimación del número de filas de empleados utilizando los dos primeros rangos puede conducir a grandes errores de estimación, ya que más de 97 por ciento de los empleados tienen salarios menores a 80 000 dólares. Por ejemplo, puede calcular 1 125 filas (12.5 por ciento de 9 000) para estimar el número de empleados que ganan entre 10 000 y 15 000 dólares, según la figura 8.4. Sin embargo, el número real de filas es mucho menor, ya que pocos empleados ganan menos de 15 000 dólares.

Debido a que los datos asimétricos pueden conducir a estimaciones pobres con el uso de los histogramas tradicionales (intervalo igual), la mayoría de los DBMS utilizan histogramas del tipo igual-altura, como se muestra en la figura 8.5. En un histograma del tipo igual-altura los rangos se determinan para que cada uno tenga aproximadamente el mismo número de filas. Por lo tanto, la anchura de los rangos varía, pero la altura es aproximadamente la misma. La mayoría de los DBMS usan histogramas del tipo igual-altura dado que los errores de estimación máximos y esperados se pueden controlar aumentando el número de rangos.

**FIGURA 8.5** Ejemplo de un histograma del tipo igual-altura para la columna salario
**TABLA 8.3**  
**Componentes típicos de un perfil de aplicación**

Tipo de aplicación	Estadísticas
Consulta	Frecuencia, distribución de los valores de parámetros
Formulario	Frecuencia de las operaciones de inserción, actualización, eliminación y recuperación
Reporte	Frecuencia, distribución de valores de parámetros

Los perfiles de las tablas se usan para estimar la medida combinada del desempeño que se presenta en la sección 8.1.2. Por ejemplo, el número de registros físicos se usa para calcular los accesos a registros físicos para extraer las filas de una tabla. La distribución de los valores de las columnas es necesaria para estimar la fracción de filas que satisfacen una condición en una consulta. Por ejemplo, para estimar la fracción de filas que satisfacen la condición *Salary > 45000*, usted tendría que sumar el número de filas de las primeras tres barras de la figura 8.4 y usar la interpolación lineal de la cuarta barra.

Algunas veces es útil almacenar datos más detallados de las columnas. Si las columnas están relacionadas, se puede caer en errores cuando se estime la fracción de filas que satisfacen las condiciones conectadas por los operadores lógicos. Por ejemplo, si las columnas salario y edad están relacionadas, la fracción de filas que satisfacen la expresión booleana *Salary > 45000 AND Age < 25*, no se puede estimar correctamente considerando la distribución del salario y la edad de forma separada. También se necesitan los datos acerca de la relación estadística entre el salario y la edad. Dado que los resúmenes acerca de las relaciones de las columnas son costosos tanto en su obtención como en almacenamiento, muchos DBMS asumen que las columnas son independientes.

### 8.2.2 Perfiles de la aplicación

Los perfiles de las aplicaciones resumen las consultas, formularios y reportes que acceden a una base de datos, como lo muestra la tabla 8.3. Para los formularios se debe especificar la frecuencia del uso del formulario para cada operación (inserción, actualización, eliminación y recuperación). Para las consultas y reportes, la distribución de los valores de los parámetros codifica el número de veces que se ejecuta la consulta/reporte con varios valores para los parámetros. Desafortunadamente, los DBMS no son tan útiles para la obtención de perfiles de aplicaciones como para los perfiles de tablas. El diseñador de bases de datos puede enfrentarse a la necesidad de escribir un software especializado o encontrar un software de terceros para obtener los perfiles de la aplicación.

**TABLA 8.4**  
**Ejemplo de perfiles de la aplicación**

Nombre de la aplicación	Tablas	Operación	Frecuencia
Consulta de inscripción	<i>Course, Offering, Enrollment</i>	Recuperar	100 por día durante el periodo de registro; 50 por día durante el periodo de altas/bajas
Formulario de registro	<i>Registration</i>	Insertar	1 000 por día durante el periodo de registro
Formulario de registro	<i>Enrollment</i>	Insertar	5 000 por día durante el periodo de registro; 1 000 por día durante el periodo de bajas/altas
Formulario de registro	<i>Registration</i>	Eliminar	100 por día durante el periodo de registro; 10 por día durante el periodo de bajas/altas
Formulario de registro	<i>Enrollment</i>	Eliminar	1 000 por día durante el periodo de registro; 500 por día durante el periodo de bajas/altas
Formulario de registro	<i>Registration, Student</i>	Recuperar	6 000 por día durante el periodo de registro; 1 500 por día durante el periodo de bajas/altas
Formulario de registro	<i>Enrollment, Course, Offering, Faculty</i>	Recuperar	6 000 por día durante el periodo de registro; 1 500 por día durante el periodo de bajas/altas
Reporte de carga de trabajo del catedrático	<i>Faculty, Course, Offering, Enrollment</i>	Recuperar	50 por día durante la ultima semana del periodo académico; de lo contrario, 10 por día; parámetros típicos: año actual y periodo académico

La tabla 8.4 ilustra los perfiles para varias aplicaciones de la base de datos de la universidad. La frecuencia de los datos se especifica como un promedio de la unidad de tiempo, como podría ser un día. Algunas veces es útil resumir las frecuencias con más detalle. La especificación de las frecuencias pico y la varianza de las frecuencias puede ayudar a evitar los problemas con el uso de los picos. Además, la importancia de las aplicaciones puede especificarse como tiempos de respuesta límite para que los diseños físicos sean parciales hacia las aplicaciones críticas.

## 8.3 Estructuras de archivos

Como se mencionó en la sección 8.1, la selección entre estructuras de archivos alternativas es una de las decisiones más importantes del diseño físico de bases de datos. Para elegir de forma inteligente, usted debe comprender las características de las estructuras de archivos disponibles. Esta sección describe las características de las estructuras de archivos disponibles en la mayoría de los DBMS.

### archivo secuencial

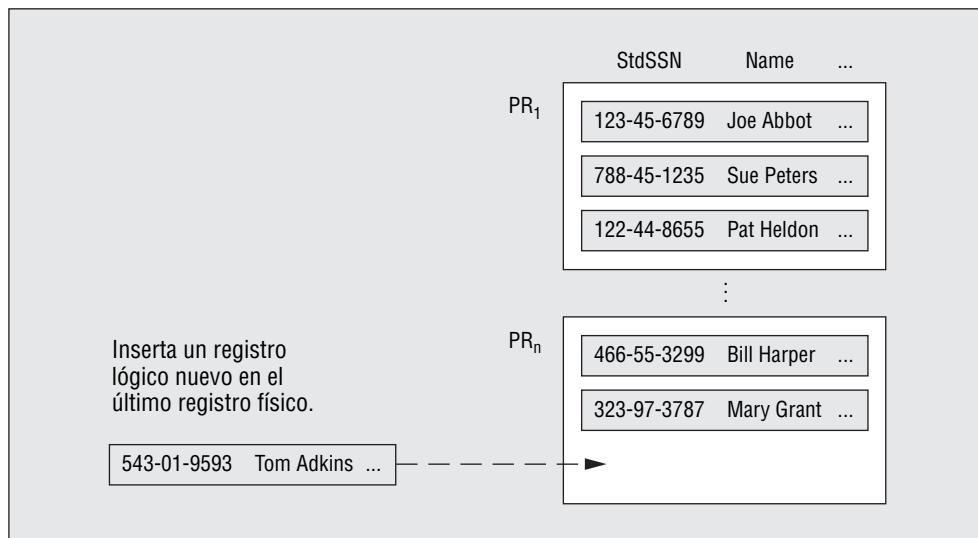
una organización de archivos simple en la que los registros se almacenan en el orden de inserción o mediante el valor de una llave. Los archivos secuenciales son más fáciles de mantener y proporcionan un buen desempeño al procesar un gran número de registros.

#### 8.3.1 Archivos secuenciales

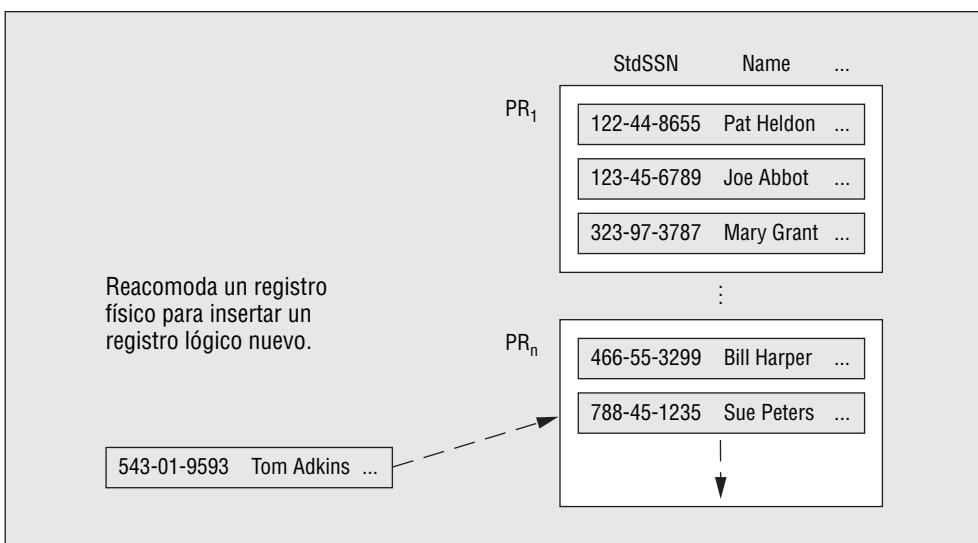
El tipo más simple de estructura de archivos almacena los registros lógicos en el orden en el que se insertaron. Los registros lógicos nuevos se agregan después del último registro físico del archivo, tal como se muestra en la figura 8.6. A menos que los registros lógicos se inserten en un orden particular y que no se eliminén, el archivo se encontrará desordenado. A los archivos desordenados algunas veces se les conoce como archivos amontonados (*heap file*) dada la carencia de orden.

La principal ventaja de los archivos secuenciales desordenados es la inserción rápida. Sin embargo, cuando se borran los registros lógicos, la inserción se vuelve más complicada. Por ejemplo, si el segundo registro lógico en PR<sub>1</sub> se elimina, existe espacio disponible en PR<sub>1</sub>. Se debe conservar una lista de espacio libre para indicar si se puede insertar un nuevo registro en el espacio libre en lugar de insertarlo en el último registro físico. De forma alternativa, los registros lógicos nuevos siempre se pueden insertar en el último registro físico. Sin embargo, es necesaria la reorganización periódica para recuperar el espacio perdido por las eliminaciones.

**FIGURA 8.6**  
Inserción de un nuevo registro lógico en un archivo secuencial desordenado



**FIGURA 8.7**  
Inserción de un nuevo registro lógico en un archivo secuencial ordenado



Debido a que en ocasiones es necesaria la recuperación ordenada, se puede preferir el uso de archivos secuenciales ordenados en lugar de archivos secuenciales desordenados. Los registros lógicos se acomodan en el orden de una llave, donde ésta puede ser cualquier columna, aunque comúnmente es la llave primaria. Los archivos secuenciales ordenados son más rápidos cuando se recuperan con el orden de la llave, ya sea el archivo completo o un subconjunto de registros. La principal desventaja para ordenar los archivos secuenciales es la baja velocidad de inserción. La figura 8.7 demuestra que los registros algunas veces deben ser reacomodados durante el proceso de inserción. El proceso de reacomodo puede incluir el movimiento de registros lógicos entre los bloques y la conservación de una lista ordenada de los registros físicos.

#### archivo hash

una estructura de archivos especializada que soporta la búsqueda por medio de una llave. Los archivos hash transforman el valor de una llave en una dirección para proporcionar un rápido acceso.

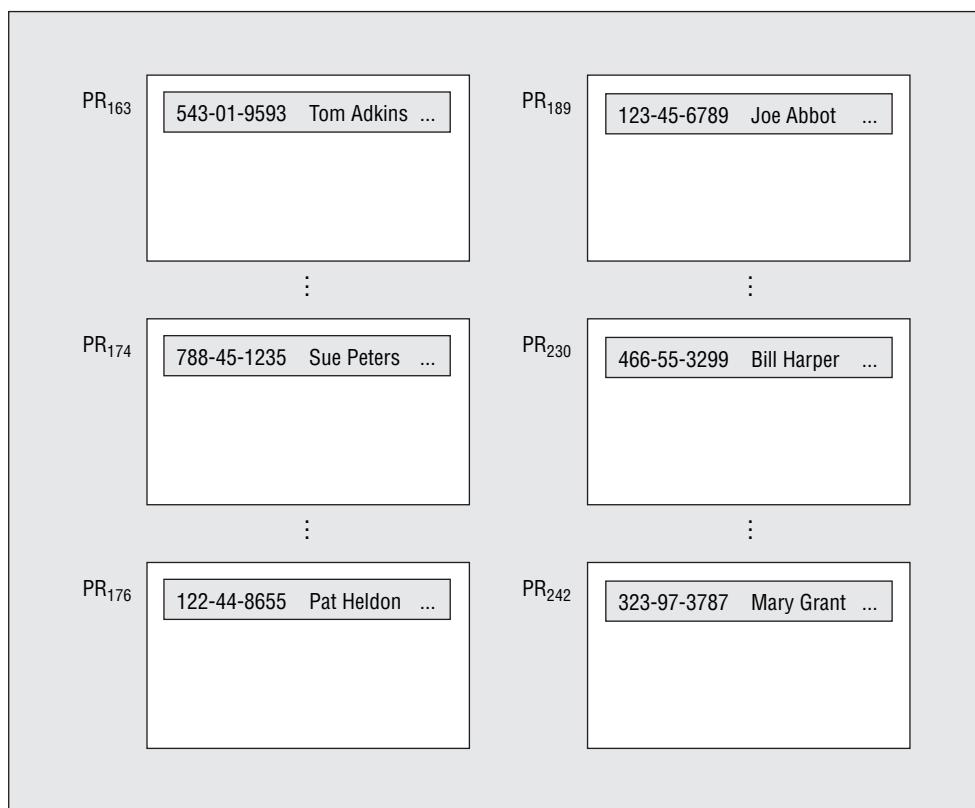
#### 8.3.2 Archivos hash

Los archivos hash, en contraste con los archivos secuenciales, soportan el acceso rápido a los registros mediante el valor de la llave primaria. La idea básica detrás de los archivos hash es una función que convierta el valor de una llave en la dirección de un registro físico. La función mod (residuo de la división) es una función hash simple. La tabla 8.5 aplica la función mod a los valores de la columna *StdSSN* de la figura 8.6. Por simplicidad, asuma que la capacidad del archivo es de 100 registros físicos. El divisor de la función mod es 97, un número primo grande

**TABLA 8.5**  
**Cálculos de la función hash para los valores de StdSSN**

StdSSN	StdSSN Mod 97	Número PR
122448655	26	176
123456789	39	189
323973787	92	242
466553299	80	230
788451235	24	174
543019593	13	163

**FIGURA 8.8**  
**Archivo hash después de las inserciones**



cercano a la capacidad del archivo. El número del registro físico es el resultado de la función hash más el número del registro físico inicial, y se asume que es 150. La figura 8.8 muestra los registros físicos seleccionados del archivo hash.

Las funciones hash pueden asignar más de una llave a la misma dirección del registro físico. Una colisión ocurre cuando dos llaves hash conducen al mismo registro físico. Una colisión no es problemática mientras que el registro físico tenga el espacio libre. Sin embargo, si el registro físico original está lleno, un procedimiento de manejo de colisiones localiza un registro físico con espacio libre. La figura 8.9 demuestra el procedimiento de la medida lineal para el manejo de colisiones. En el procedimiento de la medida lineal, se coloca un registro lógico en el siguiente registro físico disponible en caso de que su dirección original se encuentre ocupada. Para recuperar un registro mediante su llave, se busca primero la dirección original. Si no se encuentra el registro en su dirección original, se ejecuta una prueba lineal.

La existencia de colisiones marca un problema potencial de los archivos hash. Si no ocurren frecuentemente las colisiones, las inserciones y extracciones son muy rápidas. Si las colisiones ocurren con frecuencia, las inserciones y extracciones pueden ser lentas. La presencia de una colisión depende de qué tan lleno se encuentre el archivo. Por lo general, no ocurren colisiones con frecuencia si el archivo tiene menos del 70 por ciento de su capacidad. Sin embargo, conservar un archivo hash que sólo esté lleno 70 por ciento puede ser un problema si la

**FIGURA 8.9**

**Prueba lineal**  
para el manejo de  
colisiones durante  
una operación de  
inscripción

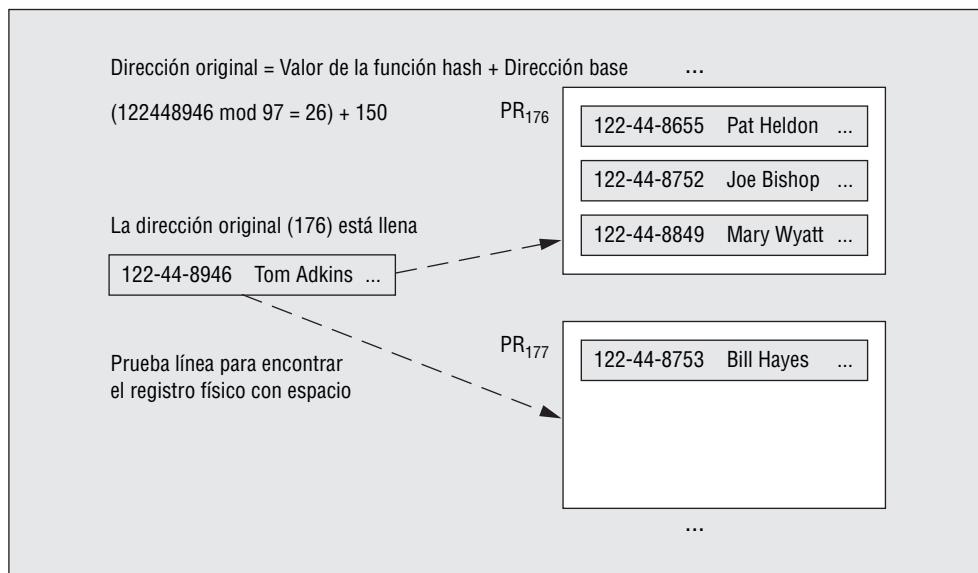


tabla crece. Si el archivo hash se llena demasiado, es necesario hacer una reorganización. Una reorganización puede consumir tiempo, dado que se localiza un archivo hash más grande y se insertan todos los registros lógicos en el archivo nuevo.

Para eliminar las reorganizaciones se han propuesto archivos hash dinámicos. En un archivo hash dinámico, la reorganización periódica nunca es necesaria y el desempeño de las búsquedas no se degrada después de muchas operaciones de inserción. Sin embargo, el promedio de accesos a los registros físicos para recuperar un registro puede ser ligeramente más alto cuando se compara con un archivo hash estático que no esté lleno. La idea básica de los archivos hash dinámicos es que el tamaño del archivo hash crezca conforme se inserten registros. Para obtener detalles de varios alcances, consulte las referencias al final de este capítulo.

Otro problema de los archivos hash es la búsqueda secuencial. Las buenas funciones hash tienden a dispersar los registros lógicos de forma uniforme entre los registros físicos. Dado los espacios que existen entre los registros físicos, la búsqueda secuencial puede examinar registros físicos vacíos. Por ejemplo, para buscar el archivo hash que se ilustra en la figura 8.8, se deben examinar 100 registros físicos, aunque sólo seis tengan datos. Incluso si el archivo hash está razonablemente lleno, los registros lógicos se dispersan entre más registros físicos que en un archivo secuencial. Por ende, cuando se realiza una búsqueda secuencial, el número de accesos a registros físicos puede ser mayor en un archivo hash que en un archivo secuencial.

**archivo Btree**  
una popular estructura  
de archivos soportada  
por la mayoría de los  
DBMS que proporciona  
un buen desempeño  
tanto en búsquedas con  
llaves como secuenciales.  
Un archivo Btree  
es un árbol multiforme,  
balanceado.

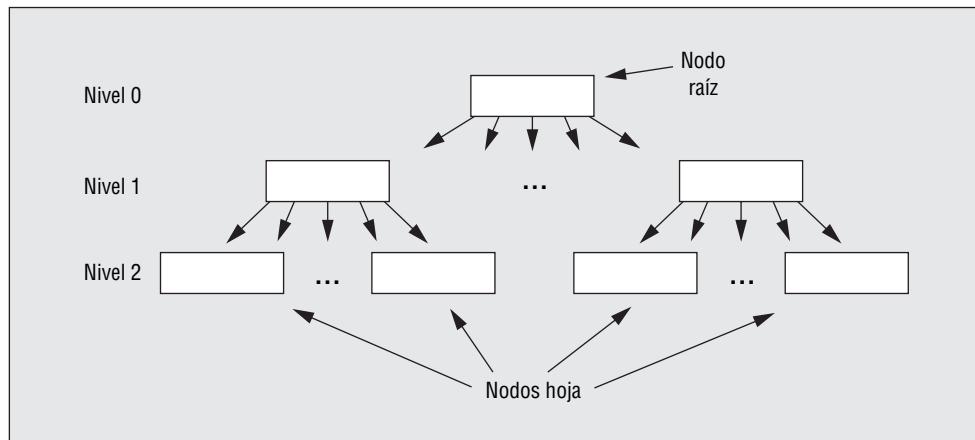
### 8.3.3 Archivos de árbol multiforme (Btrees)

Los archivos secuenciales y archivos hash proporcionan un buen desempeño en algunas operaciones pero un pobre desempeño en otras. Los archivos secuenciales se desempeñan bien en búsquedas secuenciales pero mal en búsquedas con llaves. Los archivos hash se desempeñan bien en búsquedas con llaves pero mal en búsquedas secuenciales. El árbol multiforme, o Btree, como se conoce popularmente, es una estructura de archivos ampliamente utilizada. Btree proporciona buen desempeño tanto en búsquedas secuenciales como con llaves. Esta sección describe las características de Btree, muestra ejemplos y describe el costo de las operaciones Btree.

#### Características de Btree: ¿Qué hay en un nombre?

Btree es un tipo especial de árbol como el que se ilustra en la figura 8.10. Un árbol es una estructura en la cual cada nodo tiene cuando mucho sólo a una madre, a excepción del nodo raíz o nodo superior. La estructura Btree posee varias características, descritas en la siguiente lista, lo que lo hace una estructura de archivos útil.

**FIGURA 8.10**  
Estructura de un  
Btree de nivel 3



Algunas de las características son los posibles significados de la letra *B* que forma parte del nombre.<sup>1</sup>

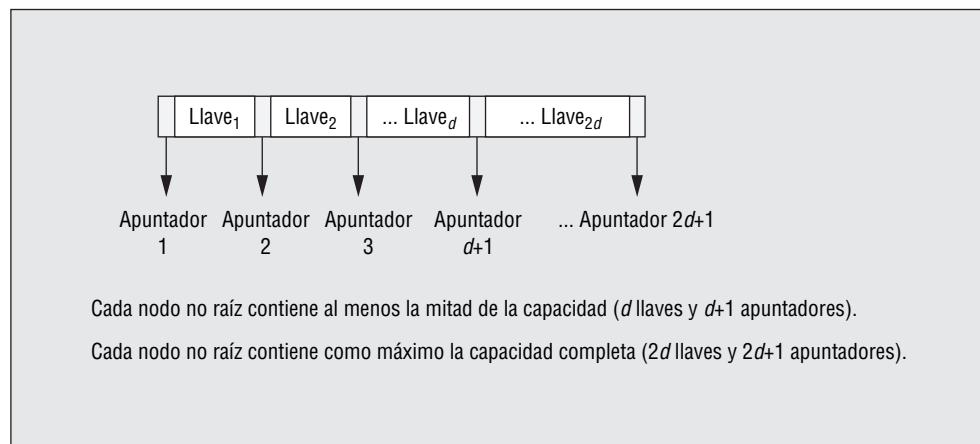
- **Balanceado:** todos los nodos hoja (nodos que no tienen hijos) residen en el mismo nivel del árbol. En la figura 8.10, todos los nodos hoja están dos niveles por debajo de la raíz. Un árbol balanceado se cerciora de que todos los nodos hoja se puedan encontrar con el mismo costo de acceso.
- **Tupido (*bushy*):** el número de ramas de un nodo es grande, tal vez entre 50 y 200 ramas. Multiforme, con más de dos, es un sinónimo de arbusto. El ancho (número de flechas a partir de un nodo) y la altura (número de nodos entre los nodos raíz y hoja) están inversamente relacionados: mientras aumente el ancho, disminuye la altura. El Btree ideal es amplio (de tipo arbusto) pero pequeño (pocos niveles).
- **Orientado a bloques (*Block-Oriented*):** cada nodo de un Btree es un bloque o un registro físico. Para buscar en un Btree, se comienza en la raíz y se sigue una ruta hasta el nodo hoja que contenga los datos que le interesan. La altura de un Btree es importante porque determina el número de accesos a registros físicos durante la búsqueda.
- **Dinámico:** la forma de un Btree cambia mientras se insertan y borran registros lógicos. Nunca es necesario hacer una reorganización periódica para un Btree. La siguiente subsección describe la división e integración de los nodos, la forma en que se modifica un Btree mientras se insertan y eliminan registros.
- **Ubicuo:** el Btree es una estructura de archivos ampliamente implementada y usada.

Antes de estudiar la naturaleza dinámica, revisemos en forma más detallada los contenidos de un nodo, como se ilustran en la figura 8.11. Cada nodo está formado por pares con un valor llave y un apuntador (dirección física del registro), ordenados por el valor de la llave. El apuntador identifica el registro físico que contiene el registro lógico con el valor de la llave. Por lo general otros datos del registro lógico, además de la llave, no residen en los nodos. Los otros datos pueden almacenarse en registros físicos separados o en los nodos hoja.

Una propiedad importante de un Btree es que cada nodo, excepto el raíz, debe estar lleno por lo menos hasta la mitad. El tamaño del registro físico es de 1 024 bytes, el tamaño de la llave es de 4 bytes y el tamaño del apuntador de 4 bytes; la máxima capacidad de un nodo es de 128 pares <llave, apuntador>. Por ende, cada nodo debe contener al menos 64 pares. Dado que generalmente el diseñador no tiene el control del tamaño del registro físico y del tamaño del

<sup>1</sup> Otro posible significado de la letra *B* es Bayer, por el nombre de su inventor, el profesor Rudolph Bayer. En una conversación privada, el profesor Bayer negó que el Btree obtuviera su nombre a partir del suyo o el de su patrón en aquellos tiempos: Boeing. Cuando se le presionó el profesor Bayer únicamente comentó que la *B* representa la *B*.

**FIGURA 8.11**  
Nodo Btree con llaves y apuntadores



apuntador, el tamaño de la llave determina el número de ramas. Los Btrees generalmente no son buenos cuando se usan tamaños de llave grandes, ya que se tienen menos ramas por cada nodo y, por lo mismo, Btrees más altos y menos eficientes.

#### *División e integración de nodos*

Las inserciones se manejan al colocar una nueva llave en un nodo que no esté lleno o bien dividiendo nodos, como se ilustra en la figura 8.12. En el Btree parcial de la figura 8.12a), cada nodo contiene un máximo de cuatro llaves. Insertar la llave con el valor de 55 en la figura 8.12b) requiere de un reacomodo en el nodo hoja de la derecha. La inserción del valor 58 en la figura 8.12c) requiere más trabajo, ya que el nodo de la derecha se encuentra lleno. Para agregar un valor nuevo, el nodo se divide en dos nodos y se coloca un valor de la llave en el nodo raíz. En la figura 8.12d), ocurre una división en dos niveles ya que ambos nodos están llenos. Cuando ocurre una división en la raíz, el árbol crece otro nivel.

Las eliminaciones se manejan quitando la llave eliminada de un nodo y reparando la estructura en caso de ser necesario, como se demuestra en la figura 8.13. Si el nodo está todavía parcialmente lleno, no se realiza ninguna acción adicional, como se demuestra en la figura 8.13b). Sin embargo, si el nodo está a menos de la mitad, debe modificarse la estructura. Si un nodo vecino contiene más de la mitad de su capacidad, se puede pedir prestada una llave, como se ilustra en la figura 8.13c). Si no se puede obtener una llave, hay que juntar los nodos, como se ilustra en la figura 8.13d).

#### *Costo de las operaciones*

La altura de un Btree es incluso más pequeña para una tabla grande cuando el factor de ramificación es grande. Una frontera superior o límite de la altura ( $h$ ) de un Btree es

$$h \leq \text{ceil}(\log_d(n + 1)/2)$$

donde

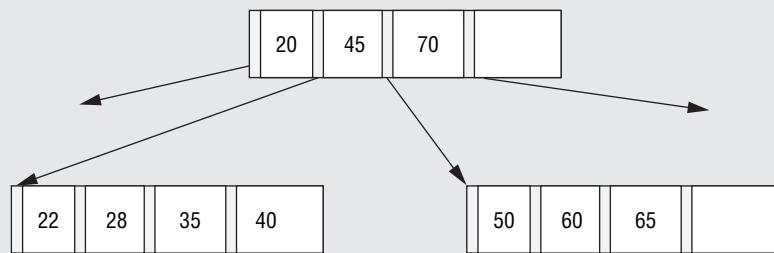
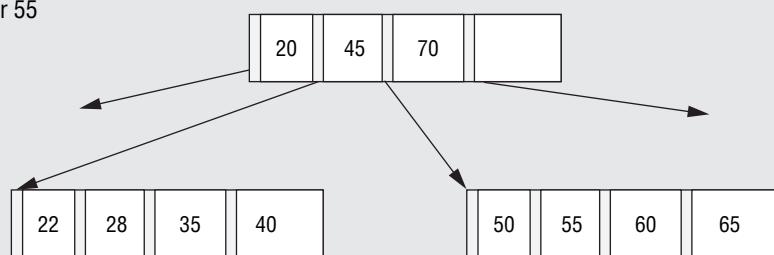
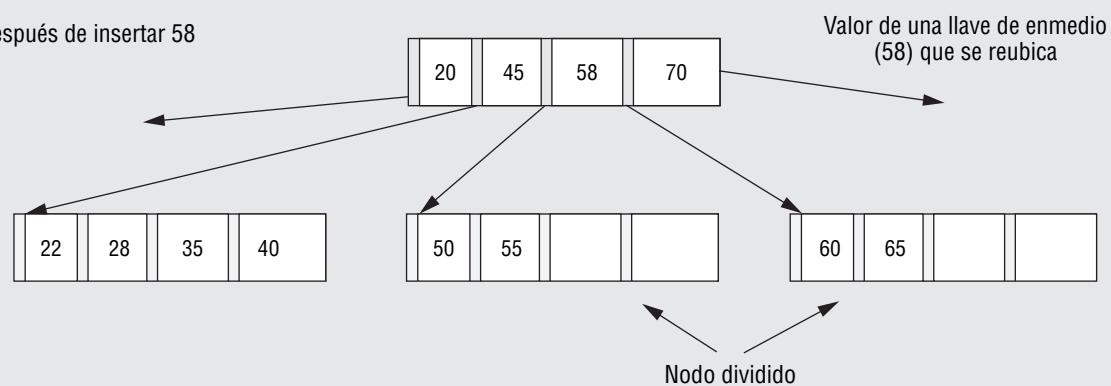
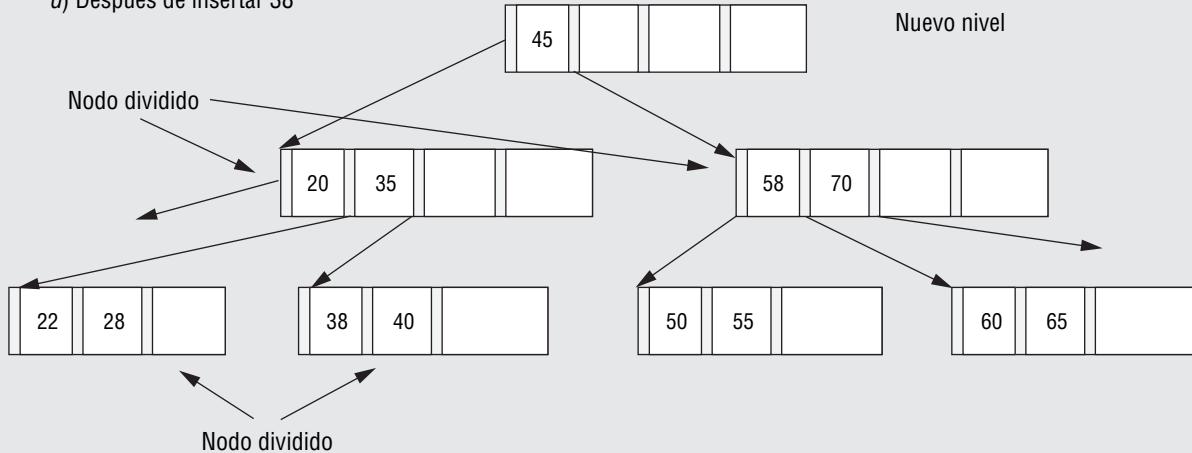
$\text{ceil}$  es la función  $\text{ceiling}$  ( $\text{ceil}(x)$  es el entero más pequeño  $\geq x$ )

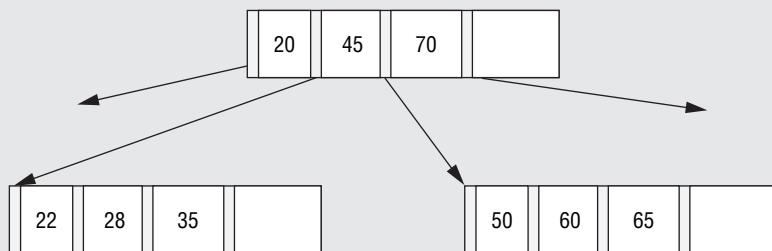
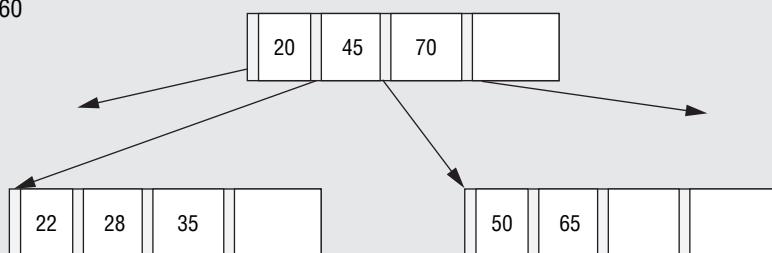
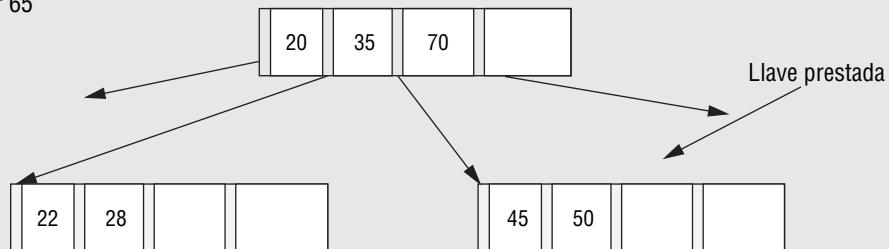
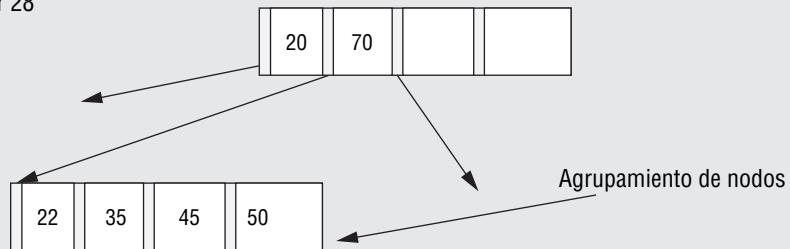
$d$  es el número mínimo de llaves de un nodo

$n$  es el número de llaves que se almacenarán en el índice

Ejemplo:  $h \leq 4$  para  $n = 1\,000\,000$  y  $d = 42$ .

En las operaciones Btree la altura domina el número de accesos a registros físicos. El costo en términos de accesos a registros físicos para encontrar una llave es menor que o igual a la altura. Si los datos de la fila no están almacenados en el árbol, se requiere de otro acceso a un registro físico para obtener los datos de la fila después de encontrar la llave. El costo de insertar alguna incluye el costo para localizar la llave más cercana más el costo de modificar los nodos. En el mejor de los casos, el costo adicional es un acceso a un registro físico para modificar el

**FIGURA 8.12** Ejemplos de inserción en Btree**a) Btree inicial****b) Despues de insertar 55****c) Despues de insertar 58****d) Despues de insertar 38**

**FIGURA 8.13** Ejemplos de eliminación de nodos**a) Btree inicial****b) Despues de borrar 60****c) Despues de borrar 65****d) Despues de borrar 28**

registro del índice y un acceso a un registro físico para escribir los datos de la fila, como se demuestra en la figura 8.12b). El peor de los casos ocurre cuando se agrega un nivel nuevo al árbol, como se ilustra en la figura 8.12d). Incluso en este último escenario aún domina la altura del árbol. Se requiere de otras operaciones  $2h$  para dividir al árbol en cada nivel.

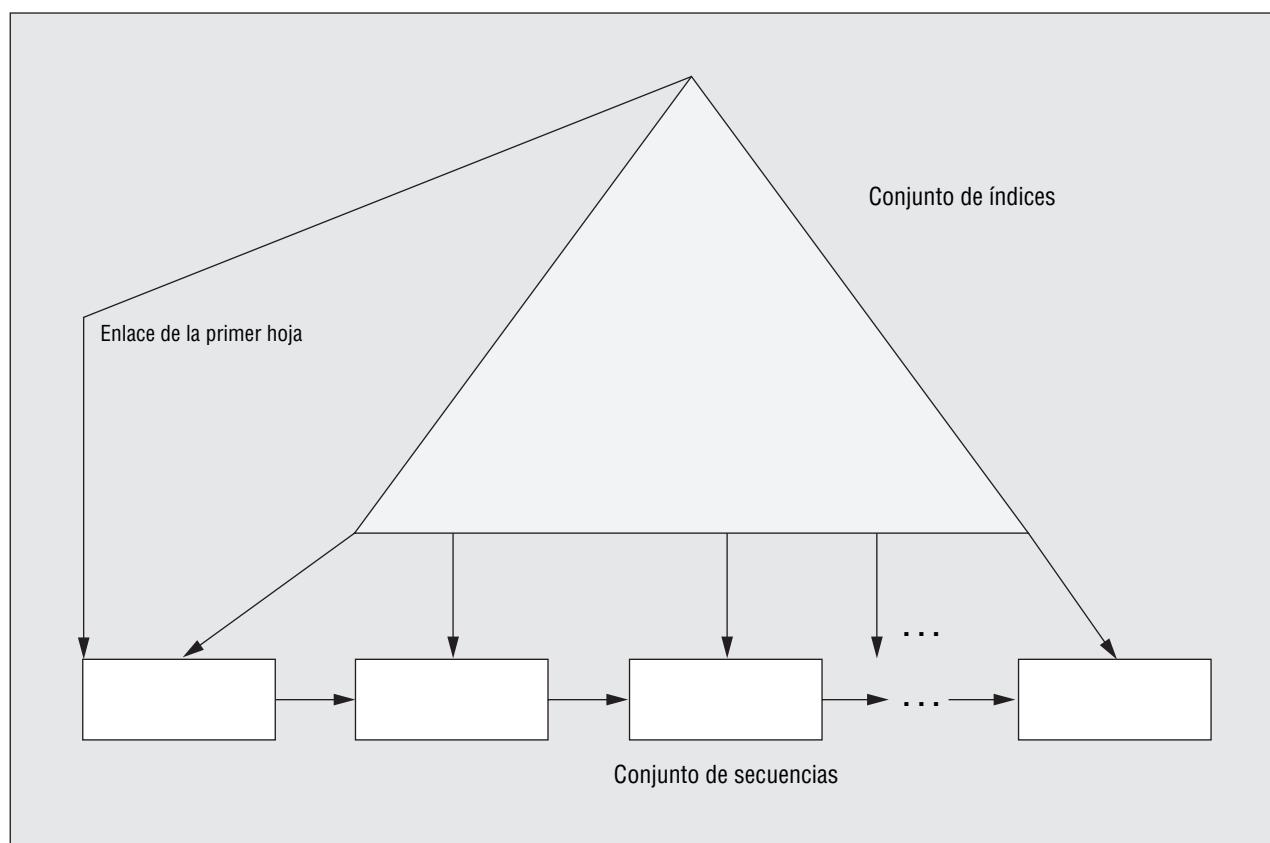
### *B+Tree*

Las búsquedas secuenciales pueden ser un problema con los Btrees. Para realizar una búsqueda de un rango, el procedimiento de búsqueda debe viajar hacia arriba y hacia abajo del árbol. Por ejemplo, para extraer las llaves que se encuentren en el rango del 28 al 60 de la figura 8.13a), el proceso de búsqueda comienza en la raíz, desciende hasta el nodo hoja de la izquierda, regresa a la raíz y después desciende hasta el nodo hoja de la derecha. Este procedimiento presenta problemas con la conservación de registros físicos en memoria. Los sistemas operativos pueden reemplazar los registros físicos si no fueron accesados recientemente. Debido a que suele pasar cierto tiempo antes de que se acceda nuevamente a un nodo madre, el sistema operativo puede reemplazarlo con otro registro físico en caso de que la memoria principal se llene. Por ende, tal vez sea necesario otro acceso a un registro físico cuando se acceda nuevamente al nodo madre.

**archivo B+Tree**  
la variante más popular de un Btree. En un B+Tree, todas las llaves se encuentran almacenadas de forma redundante en los nodos hoja. El B+Tree proporciona un mejor desempeño en las búsquedas secuenciales y de rangos.

Para asegurar que no se reemplacen los registros físicos, usualmente se implementa la variación del B+Tree. La figura 8.14 muestra las dos partes de un B+Tree. El triángulo (conjunto de índices) representa un índice normal de un Btree. La parte inferior (conjunto de secuencias) contiene los nodos hoja. Todas las llaves se ubican en los nodos hoja incluso cuando una llave aparece en el conjunto de índices. Los nodos hoja están conectados para que las búsquedas secuenciales no necesiten moverse hacia arriba del árbol. Una vez que se encuentra la llave inicial, el proceso de búsqueda accede únicamente a los nodos del conjunto de la secuencia.

**FIGURA 8.14 Estructura B+Tree**



### Coincidencia de índices

Se puede usar un Btree para almacenar todos los datos en los nodos (estructura de archivos primaria) o sólo los apuntadores hacia los registros de datos (estructura de archivos secundaria o índice). Un Btree es especialmente versátil como índice, ya que se puede usar para almacenar diversas consultas. A la determinación de si se puede usar un índice en una consulta se le conoce como coincidencia del índice. Cuando una condición en una cláusula WHERE hace referencia a una columna indexada, el DBMS debe determinar si es posible usar el índice. La complejidad de una condición determina si se puede usar un índice. Para índices de columna única, un índice coincide con una condición si la columna aparece sola, sin funciones u operadores, y el operador de comparación coincide con uno de los siguientes elementos:

=, >, <, >=, <= (pero no <>)  
 BETWEEN  
 IS NULL  
 IN <lista de valores constantes>  
 LIKE ‘Patrón’ en el cual patrón no contiene ningún metacarácter (%, \_) como la primera parte del patrón

Para los índices compuestos que involucren más de una columna, las reglas de coincidencia son más complejas y restrictivas. Los índices compuestos están ordenados de la columna más significativa (primera columna del índice) a la columna menos significativa (última columna del índice). Un índice compuesto coincide con las condiciones de acuerdo con las siguientes reglas:

- La primera columna del índice debe tener una condición de coincidencia.
- Las columnas coinciden de izquierda (más significativo) a derecha (menos significativo). La coincidencia se detiene cuando la siguiente columna del índice no coincide.
- Por lo menos una condición BETWEEN coincide. Ninguna otra condición coincide después de la condición BETWEEN.
- Por lo menos una condición IN coincide con una columna índice. Las coincidencias se detienen después de la siguiente condición. La segunda condición no puede ser IN o BETWEEN.

Para ilustrar las coincidencias con los índices, la tabla 8.6 muestra ejemplos de coincidencia entre índices y condiciones. Cuando se utiliza un índice compuesto, las condiciones pueden estar en cualquier orden. Los índices compuestos se deben usar con precaución debido a las

**TABLA 8.6**  
**Ejemplos de coincidencias con índices**

Condición	Índice	Notas de la coincidencia
C1 = 10	C1	Coincide con el índice en C1
C2 BETWEEN 10 AND 20	C2	Coincide con el índice en C2
C3 IN (10, 20)	C3	Coincide con el índice en C3
C1 < > 10	C1	No coincide con el índice en C1
C4 LIKE ‘A%’	C4	Coincide con el índice en C4
C4 LIKE ‘%A’	C4	No coincide con el índice en C4
C1 = 10 AND C2 = 5 AND C3 = 20 AND C4 = 25	(C1,C2,C3,C4)	Coincide con todas las columnas del índice
C2 = 5 AND C3 = 20 AND C1 = 10	(C1,C2,C3,C4)	Coincide con las primeras tres columnas del índice
C2 = 5 AND C4 = 22 AND C1 = 10 AND C6 = 35	(C1,C2,C3,C4)	Coincide con las primeras dos columnas del índice
C2 = 5 AND C3 = 20 AND C4 = 25	(C1,C2,C3,C4)	No coincide con ninguna columna del índice: condición perdida en C1
C1 IN (6, 8, 10) AND C2 = 5 AND C3 IN (20, 30, 40)	(C1,C2,C3,C4)	Coincide con las primeras dos columnas del índice: como máximo una coincidencia la condición IN
C2 = 5 AND C1 BETWEEN 6 AND 10	(C1,C2,C3,C4)	Coincide con la primera columna del índice: la coincidencia se detiene después de la condición BETWEEN

reglas de restricción de coincidencias. Por lo general, es mejor crear índices sobre las columnas individuales, ya que la mayoría de los DBMS pueden combinar los resultados de varios índices cuando se responde a una consulta.

### 8.3.4 Índices bitmap

Los archivos Btree y hash trabajan mejor con columnas que con valores únicos. Para las columnas que no son únicas, los nodos de los índices Btree pueden almacenar una lista de identificadores de filas en lugar del identificador de una fila única. Sin embargo, si una columna tiene pocos valores, la lista de los identificadores de filas puede ser larga.

**índices bitmap**  
estructura secundaria de archivos consistente en un valor de columna y un bitmap. Un bitmap contiene una posición de bit para cada fila de la tabla referenciada. Un índice de columna bitmap hace referencia a las filas que contienen el valor de la columna. Un índice bitmap de enlace hace referencia a las filas de una tabla hija que se une con filas de la tabla madre contenidas en la columna. Los índices bitmap funcionan correctamente para columnas estables con algunos valores típicos de tablas en un almacén de datos.

Como estructura alternativa para columnas con pocos valores, muchos DBMS soportan los índices bitmap. La figura 8.15 ilustra un índice bitmap para la tabla de ejemplo *Faculty*. Un bitmap contiene una cadena de bits (valores 0 o 1) con un bit para cada fila de una tabla. En la figura 8.15, la longitud del bitmap es de 12 posiciones, ya que existen 12 filas en la tabla de ejemplo *Faculty*. Un registro de una columna bitmap contiene un valor para la columna y un bitmap. Un valor 0 en el bitmap indica que la fila asociada no contiene el valor de la columna. Un valor 1 indica que la fila asociada tiene un valor en la columna. El DBMS proporciona una forma eficiente para convertir una posición en un bitmap que identifique las filas.

Una variante del índice de columnas bitmap es el índice bitmap de enlace. En un índice bitmap de enlace, el bitmap identifica las filas de una tabla relacionada, y no la tabla que contiene las columnas indexadas. Por ende, un índice bitmap de enlace representa un enlace precalculado de una columna en una tabla madre a las filas de una tabla hija que se enlazan con filas de la tabla madre.

Un índice bitmap de enlace se puede definir para una columna de enlace tal como *FacSSN*, o una columna que no sea de enlace, como *FacRank*. La figura 8.16 ilustra un índice bitmap de enlace para la columna *FacRank* de la tabla *Faculty* hacia las filas de la tabla *Offering*. La longitud del bitmap es de 24 bits, ya que existen 24 filas en la tabla *Offering*. Un valor de 1 en el bitmap indica que una fila madre que contenga el valor enlaza con la tabla hija en la posición especificada. Por ejemplo, un 1 en la primera posición de la fila Asst del índice bitmap de enlace significa que la fila *Faculty* con el valor Asst se enlaza con la primera fila de la tabla *Offering*.

Los índices bitmap de enlace funcionan bien para columnas estables con pocos valores. La columna *FacRank* sería atractiva para un índice bitmap de enlace porque contiene pocos valores y los miembros del profesorado no cambian de rango de forma continua. El tamaño del bitmap

**FIGURA 8.15**

Tabla Faculty de ejemplo e índice bitmap de columna en FacRank

Tabla Faculty

RowId	FacSSN	...	FacRank
1	098-55-1234		Asst
2	123-45-6789		Asst
3	456-89-1243		Assc
4	111-09-0245		Prof
5	931-99-2034		Asst
6	998-00-1245		Prof
7	287-44-3341		Assc
8	230-21-9432		Asst
9	321-44-5588		Prof
10	443-22-3356		Assc
11	559-87-3211		Prof
12	220-44-5688		Asst

Índice bitmap de columna en FacRank

FacRank	Bitmap
Asst	110010010001
Assc	001000100100
Prof	000101001010

**FIGURA 8.16**  
**Tabla Offering de ejemplo e índice bitmap de enlace en FacRank**

Tabla Offering			
RowId	OfferNo	...	FacSSN
1	1111		098-55-1234
2	1234		123-45-6789
3	1345		456-89-1243
4	1599		111-09-0245
5	1807		931-99-2034
6	1944		998-00-1245
7	2100		287-44-3341
8	2200		230-21-9432
9	2301		321-44-5588
10	2487		443-22-3356
11	2500		559-87-3211
12	2600		220-44-5688
13	2703		098-55-1234
14	2801		123-45-6789
15	2944		456-89-1243
16	3100		111-09-0245
17	3200		931-99-2034
18	3258		998-00-1245
19	3302		287-44-3341
20	3901		230-21-9432
21	4001		321-44-5588
22	4205		443-22-3356
23	4301		559-87-3211
24	4455		220-44-5688

Índice bitmap de enlace en FacRank

FacRank	Bitmap
Asst	110010010001110010010001
Assc	001000100100001000100100
Prof	000101001010000101001010

no es un punto importante ya que las técnicas de compresión pueden reducir el tamaño en forma significativa. Debido al requerimiento para las columnas estables, los índices bitmap son más comunes para las tablas de un *data warehouse*, especialmente como índices de enlace. Un *data warehouse* es una base de datos que apoya las decisiones y se usa de forma más frecuente para recuperaciones e inserciones periódicas de datos nuevos. El capítulo 16 describe el uso de índices bitmap sobre los datos de las tablas de un *data warehouse*.

### 8.3.5 Resumen de estructuras de archivos

Para ayudarle a recordar las estructuras de archivos, la tabla 8.7 resume las características principales de cada estructura. En la primera fila, los archivos hash se pueden usar para accesos secuenciales, pero quizás existan registros físicos adicionales debido a que las llaves se encuentren espaciadas entre los registros físicos. En la segunda fila, los archivos secuenciales ordenados y desordenados deben examinar el promedio de la mitad de los registros físicos (lineal). Los archivos hash examinan un número constante (generalmente cercano al 1) de registros físicos, asumiendo que el archivo no esté lleno. Los Btrees tienen costos de búsqueda logarítmicos, dada la relación entre la altura, la función de registro y las fórmulas de costo de búsquedas. Las estructuras de archivos pueden almacenar todos los datos de una tabla (estructura primaria de archivos) o almacenar únicamente los datos de la llave junto con los punteros hacia los registros de datos (estructura secundaria de archivos). Una estructura secundaria de archivos o indexada

**TABLA 8.7**  
Resumen de estructuras de archivos

	Desordenada	Ordenada	Hash	B+tree	Bitmap
Búsqueda secuencial	Y	Y	PRs adicionales	Y	N
Búsqueda por llaves	Lineal	Lineal	Tiempo constante	Logarítmica	Y
Búsqueda de rangos	N	Y	N	Y	Y
Uso	Sólo primaria	Sólo primaria	Primaria o secundaria	Primaria o secundaria	Sólo secundaria

proporciona una ruta alternativa hacia los datos. Un índice bitmap soporta las búsquedas de rangos realizando operaciones de enlace en los bitmaps para cada valor de columna dentro del rango.

## 8.4 Optimización de consultas

En la mayoría de los DBMS, usted no tendrá la habilidad de escoger la implementación de las consultas sobre la base de datos física. El componente de optimización de consultas asume esta responsabilidad. Su productividad aumenta ya que usted no necesita tomar estas tediosas decisiones. Sin embargo, algunas veces puede mejorar estas decisiones de optimización si comprende los principios del proceso de optimización. Para proporcionarle un entendimiento del proceso de optimización, esta sección describe las tareas realizadas y describe los consejos para mejorar las decisiones.

### 8.4.1 Tareas de traducción

Cuando usted introduce una sentencia SQL para su ejecución, el componente de optimización de consultas traduce su consulta en cuatro fases, como se muestra en la figura 8.17. La primera y cuarta fases son comunes para cualquier proceso de traducción de algún lenguaje de cómputo. La segunda fase tiene algunos aspectos únicos. La tercera fase es única para la traducción de lenguajes de bases de datos.

#### *Sintaxis y análisis de semántica*

La primera fase analiza una consulta para encontrar errores de sintaxis y semántica simple. Los errores de sintaxis involucran el mal uso de palabras clave, como si la palabra reservada FROM estuviera mal escrita en el ejemplo 8.1. Los errores de semántica incluyen el mal uso de columnas y tablas. El compilador del lenguaje de datos puede detectar únicamente errores simples de semántica que involucren tipos de datos incompatibles. Por ejemplo, una condición WHERE que compare la columna *CourseNo* con la columna *FacSalary* genera un error semántico ya que estas columnas tienen tipos de datos incompatibles. Para encontrar errores de semántica, el DBMS usa las definiciones de tablas, columnas y relaciones, tal como se almacenan en el diccionario de datos.

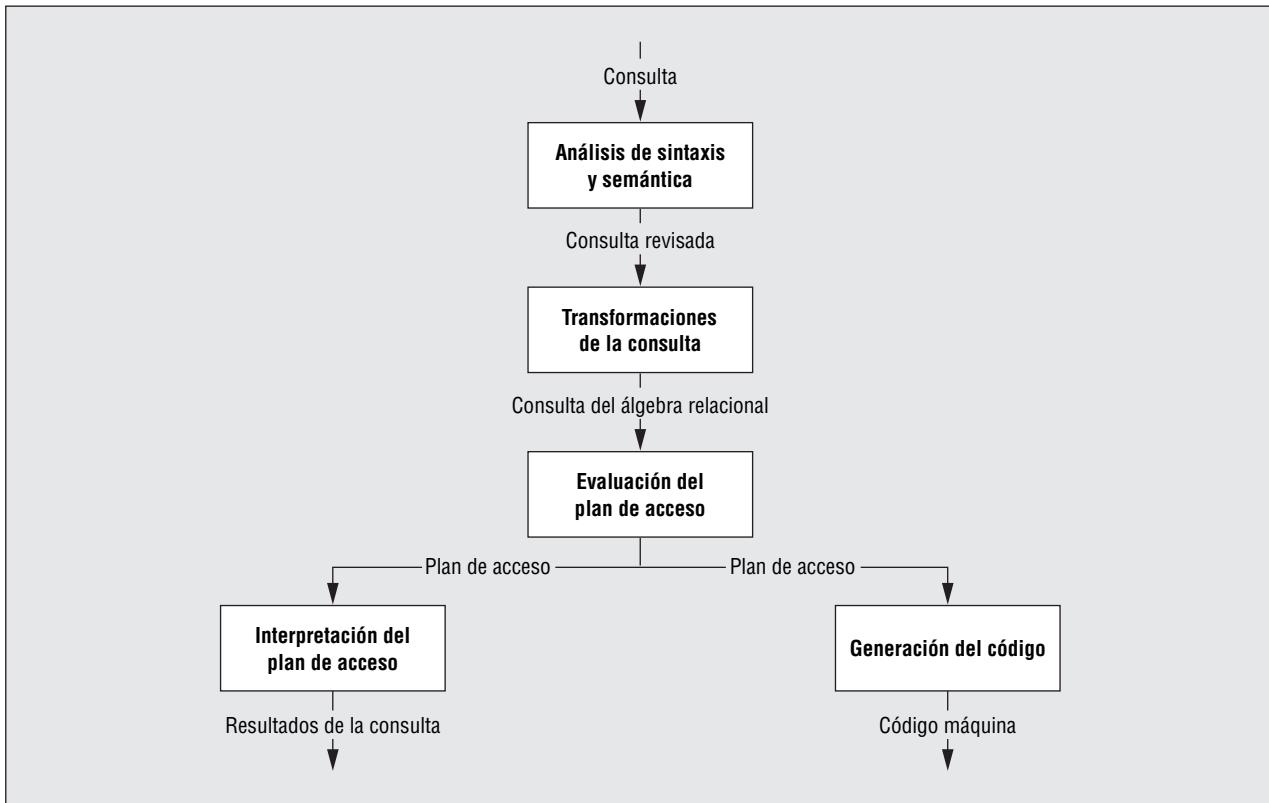
#### EJEMPLO 8.1 (Oracle)

##### Enlace de tres tablas

```
SELECT FacName, CourseNo, Enrollment.OfferNo, EnrGrade
      FROM Enrollment, Offering, Faculty
     WHERE CourseNo LIKE 'IS%' AND OffYear = 2005
           AND OffTerm = 'FALL'
           AND Enrollment.OfferNo = Offering.OfferNo
           AND Faculty.FacSSN = Offering.FacSSN
```

#### *Transformación de consultas*

La segunda fase transforma una consulta en un formato simplificado y estandarizado. Como con la optimización de los compiladores de lenguajes de programación, los traductores del lenguaje de base de datos pueden eliminar las partes redundantes de una expresión lógica. Por ejemplo, la expresión lógica (*OffYear = 2006 AND OffTerm = 'WINTER'*) OR (*OffYear = 2006 AND OffTerm = 'SPRING'*) se puede simplificar en *OffYear = 2006 AND (OffTerm = 'WINTER' OR OffTerm = 'SPRING')*.

**FIGURA 8.17** Tareas en la traducción del lenguaje de bases de datos

*OR OffTerm = 'SPRING'). La simplificación delenlace (*join*) es única para los lenguajes de bases de datos. Por ejemplo, si el ejemplo 8.1 tuviera un enlace con la tabla *Student*, esta tabla se podría eliminar si no se usaran columnas o condiciones que incluyan la tabla *Student* en la consulta.*

El formato estandarizado generalmente se basa en el álgebra relacional. Las operaciones del álgebra relacional se reacomodan de tal forma que la consulta se pueda ejecutar de manera más rápida. Las operaciones típicas de reacomodo se describen más adelante. Dado que el componente de optimización de consultas lleva a cabo este reacomodo, usted no tiene que ser cuidadoso al escribir su consulta en forma eficiente.

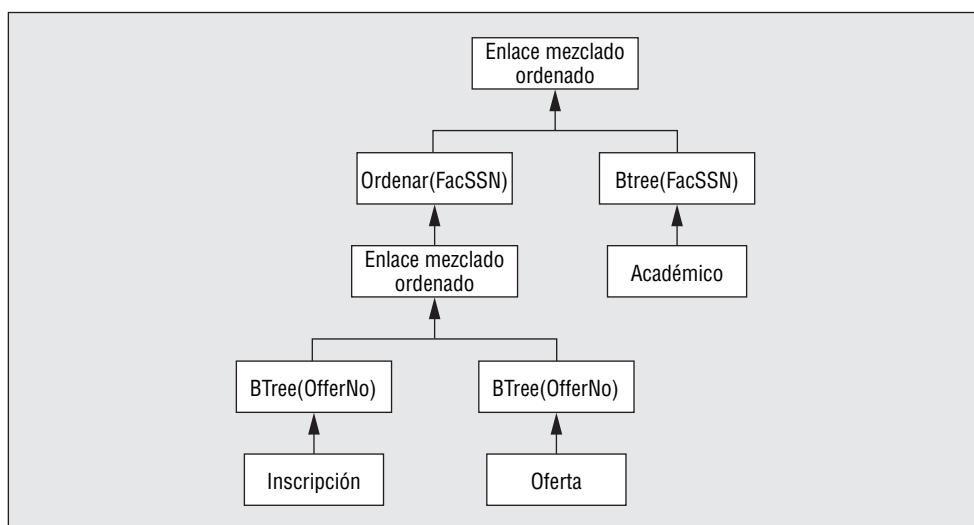
- Las operaciones de restricción se combinan para que se puedan probar de manera conjunta.
- Las operaciones de proyección y restricción se mueven para que estén antes de las operaciones de enlace (*join*) para eliminar las columnas y renglones innecesarios antes de las operaciones de enlace costosas.
- Las operaciones de productos cruz se transforman en operaciones de enlace si una condición existe en la cláusula WHERE.

#### Evaluación del plan de accesos

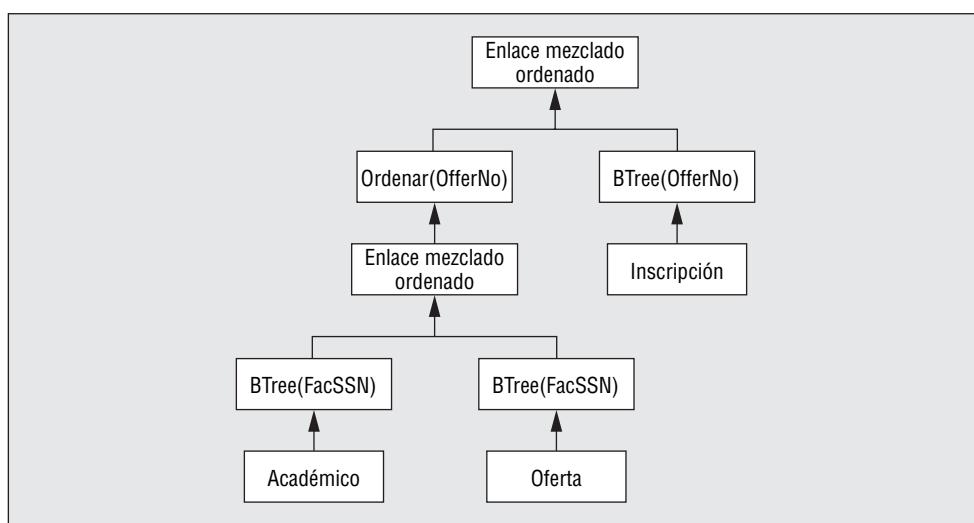
**plan de accesos**  
un árbol que codifica las decisiones acerca de las estructuras de archivos para acceder a las tablas individuales, el orden de enlace de las tablas y el algoritmo de las tablas de enlace.

La tercera fase genera un plan de acceso para implementar la consulta reacomodada del álgebra relacional. Un plan de accesos indica la implementación de una consulta como operaciones en archivos, tal como se ilustra en la figura 8.18. En un plan de accesos, los nodos hoja son tablas individuales de la consulta y las flechas apuntan hacia arriba para indicar el flujo de datos. Los nodos que están sobre los nodos hoja señalan las decisiones sobre el acceso a las tablas individuales. En la figura 8.18, los índices Btree se usan para acceder a las tablas individuales. El primer enlace combina las tablas *Enrollment* y *Offering*. Las estructuras de archivos Btree proporcionan la clasificación requerida por el algoritmo de *merge join*. El segundo enlace (*join*)

**FIGURA 8.18**  
Plan de acceso para el ejemplo 8.1



**FIGURA 8.19**  
Plan de acceso alternativo para el ejemplo 8.1



combina el resultado del primer *join* con la tabla *Faculty*. El resultado intermedio debe estar ordenado en *FacSSN* antes de que se pueda usar el algoritmo de *merge join*.

El componente de optimización de consultas evalúa un gran número de planes de acceso. Los planes de acceso varían debido al orden de los *joins*, estructuras de archivo y algoritmos de *join*. Por ejemplo, la figura 8.19 muestra una variación del plan de acceso de la figura 8.18, en el que el orden del *join* está modificado. Para las estructuras de archivos, algunos componentes de optimización pueden considerar operaciones de conjuntos (intersección para las condiciones conectadas por un AND y las condiciones de enlace conectadas por un OR) para combinar los resultados de múltiples índices sobre la misma tabla. El componente de optimización de consultas puede evaluar muchos más planes de acceso que los que un programador de bases de datos experimentado pudiese considerar. La evaluación de los planes de acceso puede involucrar una significativa cantidad de tiempo cuando la consulta contiene más de cuatro tablas.

La mayoría de los componentes de optimización utilizan un pequeño conjunto de algoritmos de *join*. La tabla 8.8 resume los algoritmos de *join* más comunes utilizados por los componentes de optimización. Para cada operación de *join* en una consulta, el componente de optimización considera cada algoritmo de *join* soportado. Para los *join* anidados y los algoritmos híbridos, el componente de optimización también debe seleccionar las tablas externa e interna. Todos los algoritmos, a excepción del enlace de estrella (*star join*), involucran dos tablas al mismo tiempo. El enlace estrella puede combinar cualquier número de tablas que coincidan con el patrón de

**TABLA 8.8** Resumen de algoritmos de *join* comunes

Algoritmo	Requerimientos	Cuándo usarlo
Ciclos anidados	Seleccione la tabla externa e interna; se puede usar para todos los <i>joins</i>	Apropiado cuando existen pocas filas en la tabla externa o cuando todas las páginas de la tabla interna caben en la memoria. Un índice en la llave foránea de la columna de <i>join</i> permite un uso eficiente del algoritmo de ciclos anidados cuando existen condiciones de restricción en la tabla madre
Mezcla ordenada ( <i>sort merge</i> )	Ambas tablas deben estar ordenadas (o usar un índice) en las columnas de <i>join</i> ; sólo se usa para los <i>equi-joins</i>	Apropiado si el costo del ordenamiento es pequeño o si existe un índice de <i>joins</i> agrupados
Enlace híbrido ( <i>hybrid join</i> )	Combinación de la mezcla ordenada y de los ciclos anidados; la tabla externa debe estar ordenada (o usar una columna índice de <i>join</i> ); la tabla interna debe tener un índice en la columna de <i>join</i> ; sólo se usa para los <i>equi-joins</i>	Se desempeña mejor que la mezcla ordenada cuando existe un índice sin agrupar (vea la siguiente sección) en la columna de <i>join</i> de la tabla interna
Enlace hash ( <i>hash join</i> )	Archivo hash interno construido para ambas tablas; sólo se usa para los <i>equi-joins</i>	Un enlace hash es mejor que un <i>sort merge</i> cuando las tablas no están ordenadas o no existen índices
Enlace de estrella ( <i>star join</i> )	Enlace de múltiples tablas en el cual existe una tabla hija relacionada con múltiples tablas madre con relaciones 1-M; se requiere un índice de <i>join</i> bitmap en cada tabla madre; sólo se usa para <i>equi-joins</i>	Es el mejor algoritmo de <i>join</i> para las tablas que coinciden con el patrón de estrella con índices de <i>join</i> de bitmap en especial cuando existen condiciones altamente selectivas en las tablas madre; ampliamente usado para optimizar las consultas de un <i>data warehouse</i> (vea el capítulo 16)

estrella (una tabla hija rodeada por tablas madre con relaciones 1-M). El algoritmo de ciclos anidados se puede usar con cualquier operación de enlace, no sólo con una operación *equi-join*.

El componente de optimización de consultas usa fórmulas de costos para evaluar los planes de acceso. Cada operación en un plan de acceso tiene una fórmula de costo correspondiente que estima los accesos de registros físicos y operaciones del CPU. Las fórmulas de costos usan los perfiles de las tablas para estimar el número de filas de un resultado. Por ejemplo, el número de filas que se generan de una condición WHERE se puede estimar usando los datos de distribución, tales como un histograma. El componente de optimización de consultas selecciona el plan de acceso con el menor costo.

#### Ejecución del plan de acceso

La última fase ejecuta el plan de acceso seleccionado. El componente de optimización de consultas genera el código máquina o interpreta el plan de acceso. La ejecución del código máquina genera una respuesta más rápida que la interpretación del plan de acceso. Sin embargo, la mayoría de los DBMS interpretan los planes de acceso debido a la amplia variedad de hardware soportado. La diferencia en el desempeño entre la interpretación y la ejecución del código máquina generalmente no es significativa para la mayoría de los usuarios.

#### 8.4.2 Mejoras en las decisiones de optimización

Aunque el componente de optimización de consultas realiza su función de forma automática, el administrador de bases de datos también debe desempeñar un rol. El administrador de bases de datos debe revisar los planes de acceso de consultas y actualizaciones que se desempeñen mal. Los DBMS corporativos típicamente proporcionan despliegues gráficos de los planes de acceso para facilitar la revisión. Los despliegues gráficos son esenciales ya que los despliegues en texto de relaciones jerárquicas son difíciles de leer.

Para mejorar las malas decisiones de los planes de accesos, algunos DBMS corporativos permiten incluir pistas que influyan en la selección de los planes de acceso. Por ejemplo, Oracle proporciona sugerencias para seleccionar la meta de optimización, las estructuras de archivos para acceder a las tablas individuales, el algoritmo y el orden del *join*. Las sugerencias deben usarse con precaución ya que pueden invalidar el juicio del optimizador. Las propuestas acerca de los

algoritmos de *join* y el orden de los *join* son especialmente problemáticos debido a lo sutil de estas decisiones. Anular el juicio del optimizador sólo debe hacerse como último recurso después de determinar la causa del mal desempeño. En muchos casos, el administrador de bases de datos puede reparar problemas con deficiencias en el perfil de tablas y en el estilo de codificación de consultas para mejorar el desempeño en lugar de anular el juicio del optimizador.

### *Deficiencias del perfil de tablas*

El componente de optimización de consultas necesita estadísticas detalladas y actualizadas para evaluar los planes de acceso. La mayoría de los DBMS proporciona control sobre el nivel de detalle de las estadísticas y su actualización. Algunos DBMS incluso permiten el muestreo dinámico de la base de datos durante el tiempo de optimización, pero normalmente este nivel de actualización de datos no es necesario.

Si las estadísticas no se recopilan para una columna, la mayoría de los DBMS utilizan el valor uniforme que se asume para estimar el número de filas. El uso del valor uniforme que se asume generalmente lleva a un acceso secuencial de archivos en lugar de un acceso con un Btree, esto en el caso de que la columna tenga suficientes inclinaciones en sus valores. Por ejemplo, considere una consulta para listar a los empleados con salarios mayores a 100 000 dólares. Si el rango del salario va de 10 000 a 2 000 000 dólares, cerca del 95 por ciento de la tabla de empleados deberá satisfacer esta condición utilizando el valor uniforme que se asume. Para la mayoría de las compañías, sin embargo, pocos empleados tendrán un salario mayor a 100 000. Utilizando la estimación del valor uniforme que se asume, el optimizador escogerá un archivo secuencial en vez de un Btree para acceder a la tabla de empleados. La estimación no mejorará mucho utilizando un histograma de igual intervalo debido a la inclinación extrema de los valores de los salarios.

Un histograma de la misma altura proporcionará mucho mejores estimaciones. Para mejorar el cálculo utilizando un histograma de la misma altura, debe aumentar el número de rangos. Por ejemplo, con 10 rangos el error máximo es de alrededor de 10 por ciento y el error esperado es de cerca de 5 por ciento. Para disminuir las estimaciones de los errores máximo y esperado en 50 por ciento, se debe duplicar el número de rangos. Un administrador de bases de datos debe aumentar el número de rangos si los errores estimados para el número de filas ocasionan selecciones inadecuadas para acceder a las tablas individuales.

Una sugerencia puede ser útil para condiciones que incluyan valores de parámetros. Si el administrador de bases de datos sabe que los típicos valores de parámetros generan un conjunto de pocas filas, se puede utilizar una sugerencia para obligar al componente de optimización a que use un índice.

Además de detallar las estadísticas de columnas individuales, un componente de optimización necesita en ocasiones estadísticas detalladas de las combinaciones de columnas. Si una combinación de columnas aparece en la cláusula WHERE de una consulta, las estadísticas de la combinación de columnas son importantes cuando las columnas no son independientes. Por ejemplo, los salarios de los empleados y sus puestos están generalmente relacionados. Una cláusula WHERE con las dos columnas del tipo *EmpPosition = 'Janitor' AND Salary > 50000* probablemente tenga pocas filas que satisfagan ambas condiciones. Un componente de optimización que desconozca la relación entre estas columnas probablemente sobreestimará el número de filas del resultado.

La mayoría de los componentes de optimización asumen que las combinaciones de columnas son estadísticamente independientes para simplificar la estimación del número de filas. Desafortunadamente, pocos DBMS conservan estadísticas de las combinaciones de columnas. Si un DBMS no conserva las estadísticas de las combinaciones de columnas, el diseñador de bases de datos pudiera utilizar las sugerencias para anular el juicio del DBMS cuando una condición de *join* en la cláusula WHERE genere pocas filas. El uso de una sugerencia puede obligar al componente de optimización a combinar índices cuando acceda a una tabla en vez de usar un escaneo secuencial de la tabla.

### *Prácticas en la codificación de consultas*

Las consultas mal escritas pueden ocasionar una ejecución lenta de las mismas. El administrador de bases de datos debe revisar las consultas que se desempeñen mal buscando las prácticas de

**TABLA 8.9** Resumen de las prácticas de codificación

Práctica de codificación	Recomendación	Elemento de desempeño
Funciones en las columnas en condiciones	Evite las funciones en columnas	Elimina la posibilidad del uso de un índice
Conversiones de tipo implícito	Use constantes con tipos de datos que coincidan con las columnas correspondientes	Elimina la posibilidad del uso de un índice
Operaciones de enlace adicionales	Elimine operaciones de enlace innecesarias buscando las tablas que no involucren condiciones o columnas	El tiempo de ejecución se determina principalmente por el número de operaciones de enlace
Condiciones en columnas de enlace	Las condiciones sobre las columnas de enlace deben usar la tabla madre y no la tabla hija	La reducción del número de filas de la tabla madre disminuirá el tiempo de ejecución de las operaciones de enlace
Condiciones de las filas en la cláusula HAVING	Mueva las condiciones de las filas de la cláusula HAVING a la cláusula WHERE	Las condiciones sobre las filas en la cláusula WHERE permiten la reducción en el tamaño del resultado intermedio
Consultas anidadas del tipo II con agrupación (capítulo 9)	Convierta las consultas anidadas del tipo II en consultas separadas	Los componentes de optimización de consultas generalmente no consideran maneras eficientes para implementar las consultas anidadas del tipo II
Consultas usando vistas complejas (capítulo 10)	Reescriba las consultas que usan vistas complejas para eliminar las referencias a vistas	Se puede ejecutar una consulta adicional
Reutilización de consultas (capítulo 11)	Asegúrese de que las consultas de un procedimiento almacenado se usen sólo una vez	El uso repetitivo involucra una sobrecarga considerable para las consultas complejas

codificación que ocasionen su lento desempeño. El resto de esta subsección explica las prácticas de codificación que ocasionan el lento desempeño de las consultas. La tabla 8.9 proporciona un resumen adecuado de las prácticas de codificación.

- Usted no debe usar funciones en columnas que tengan índices, ya que las funciones eliminan la oportunidad de utilizar un índice. Debe poner especial atención en las conversiones de tipo implícito, incluso si no usa una función. Una conversión de tipo implícito ocurre cuando no coinciden el tipo de dato de una columna y el valor constante asociado. Por ejemplo, la condición *OffYear = '2005'* ocasiona una conversión implícita de la columna *OffYear* en un tipo de dato carácter. La conversión elimina la posibilidad de usar un índice en *OffYear*.
- Las consultas con operaciones adicionales de *join* disminuirán el desempeño, como se indica en la sección 8.4.1, subsección Transformación de consultas. La velocidad de ejecución de una consulta se determina principalmente por el número de operaciones de *join*, por lo que la eliminación de operaciones de *join* innecesarias puede disminuir significativamente el tiempo de ejecución.
- Para consultas que involucren relaciones 1-M en las que exista una condición en la columna de *join*, debe hacer la condición en la tabla madre en lugar de la tabla hija. La condición de la tabla madre puede reducir de forma significativa el esfuerzo al enlazar las tablas.
- Para consultas que involucren la cláusula HAVING, elimine las condiciones que no involucren funciones agregadas. Las condiciones que involucran comparaciones sencillas de las columnas de la cláusula GROUP BY pertenecen a la cláusula WHERE, no a la cláusula HAVING. Si mueve estas condiciones a la cláusula WHERE se eliminarán las filas más pronto, por lo tanto, la ejecución será más rápida.
- Debe evitar las consultas anidadas del tipo II (vea el capítulo 9), en especial cuando la consulta anidada lleve a cabo la agrupación con cálculos agregados. Muchos DBMS son lentos porque los componentes de optimización de consultas generalmente no consideran formas eficientes para implementar las consultas anidadas del tipo II. Usted puede mejorar la velocidad de ejecución de una consulta reemplazando la consulta anidada del tipo II con una consulta separada.

### ligadura de consultas (query binding)

la asociación de un plan de acceso con una sentencia SQL. La ligadura puede reducir el tiempo de ejecución para consultas complejas ya que las fases consumidoras de tiempo del proceso de traducción no se llevan a cabo hasta que ocurre la ligadura inicial.

- Las consultas con vistas complejas pueden conducir a un lento desempeño debido a que se puede ejecutar una consulta adicional. El capítulo 10 describe el procesamiento de vistas con algunas guías para limitar la complejidad de las vistas.
- El proceso de optimización puede consumir tiempo, en especial para las consultas que contienen más de cuatro tablas. Para reducir el tiempo de optimización, la mayoría de los DBMS guardan planes de acceso para evitar las fases que consumen tiempo del proceso de traducción. La ligadura de consultas es el proceso de asociar una consulta con un plan de acceso. La mayoría de los DBMS hacen las ligaduras de forma automática si una consulta cambia o si lo hace la base de datos (estructuras de archivos, perfiles de tablas, tipos de datos, etc.). El capítulo 11 describe la ligadura de consultas para sentencias SQL dinámicas dentro de un programa de cómputo.

## 8.5 Selección de índices

### índice

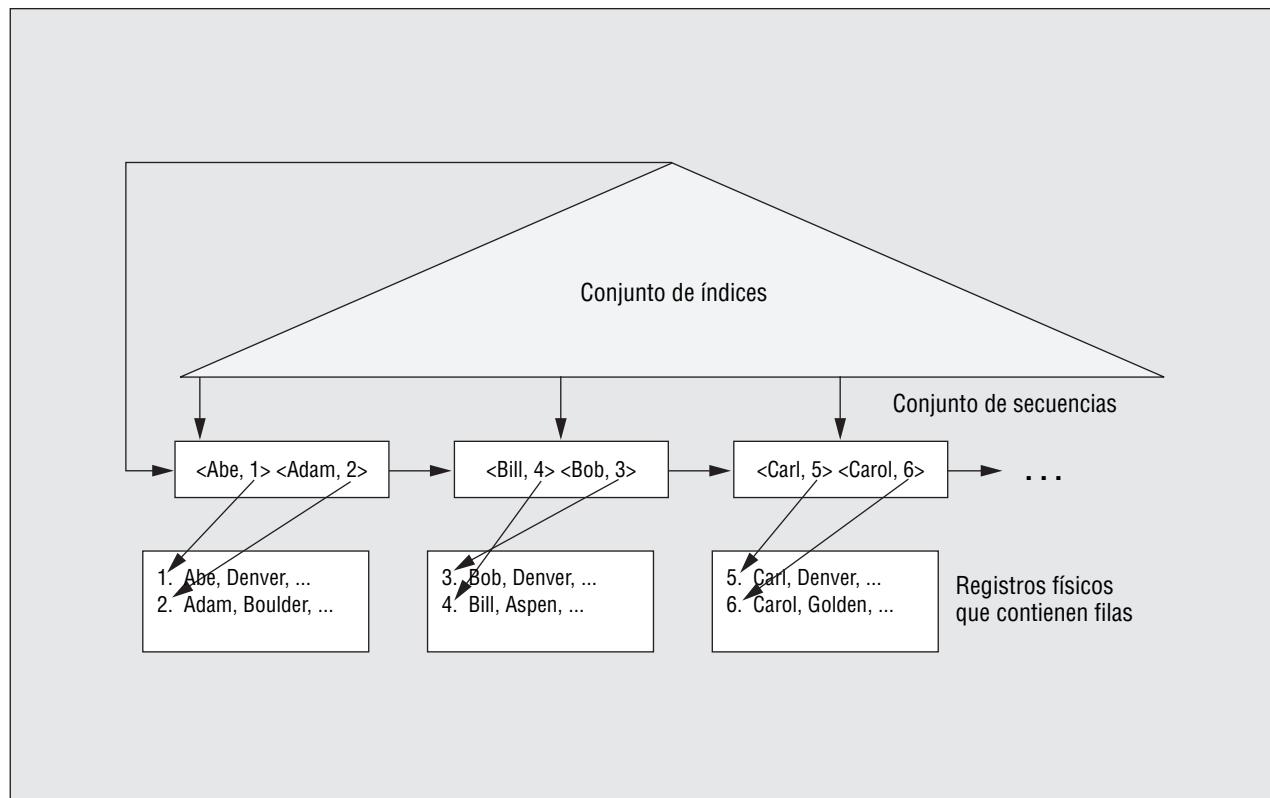
una estructura secundaria de archivos que proporciona una ruta alternativa hacia los datos. En un índice agrupado, el orden de los registros de datos es cercano al orden del índice. En un índice desagrupado, el orden de los registros de los datos no está relacionado al orden del índice.

La selección de índices es la decisión más importante para el diseñador de la base de datos física. Sin embargo, también puede ser una de las decisiones más difíciles. Como diseñador, usted necesita entender la dificultad de la selección de índices y las limitaciones de llevar a cabo la selección de índices sin una herramienta automatizada. Esta sección le ayuda a obtener este conocimiento definiendo el problema de selección de índices, describiendo las ventajas y desventajas en la selección de índices y presentando las reglas de selección de índices para bases de datos de tamaño moderado.

### 8.5.1 Definición del problema

La selección de índices incluye dos tipos de índices: agrupados y desagrupados. En un índice agrupado, el orden de las filas es cercano al orden del índice. Cercano significa que los registros

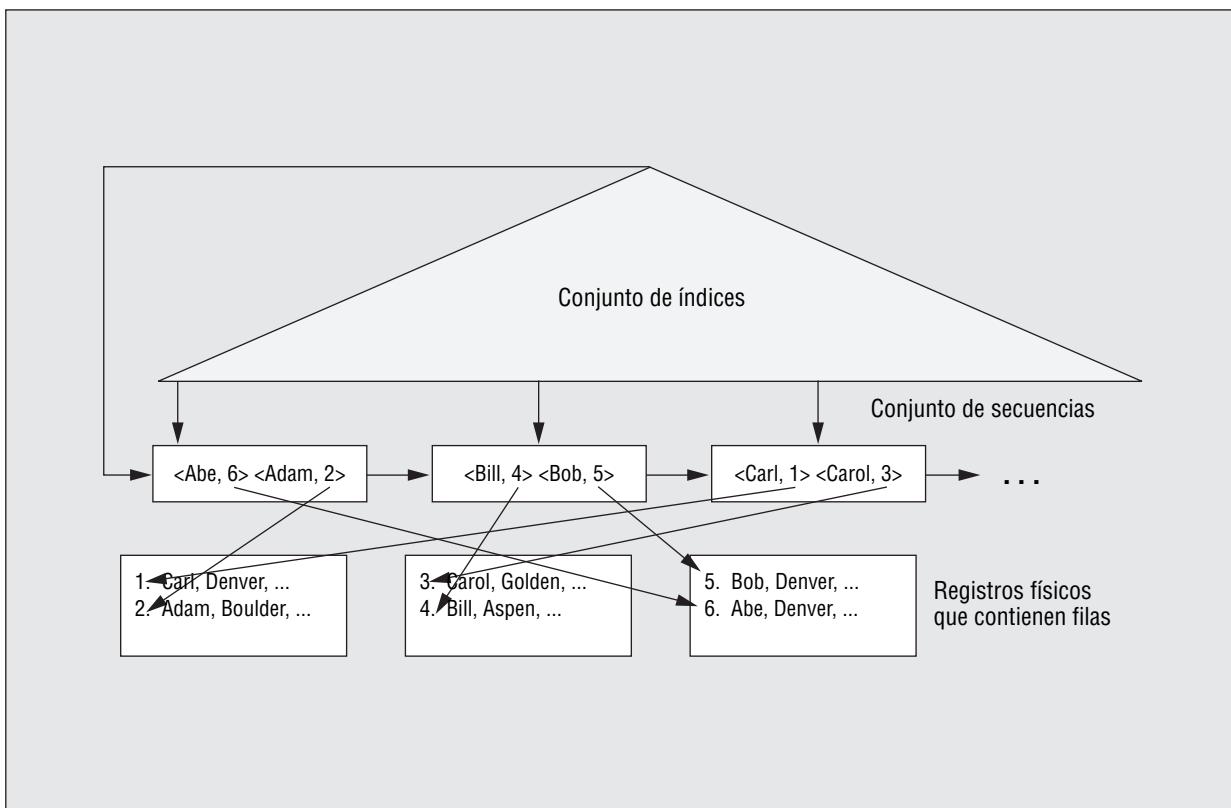
**FIGURA 8.20 Ejemplo de índice agrupado**



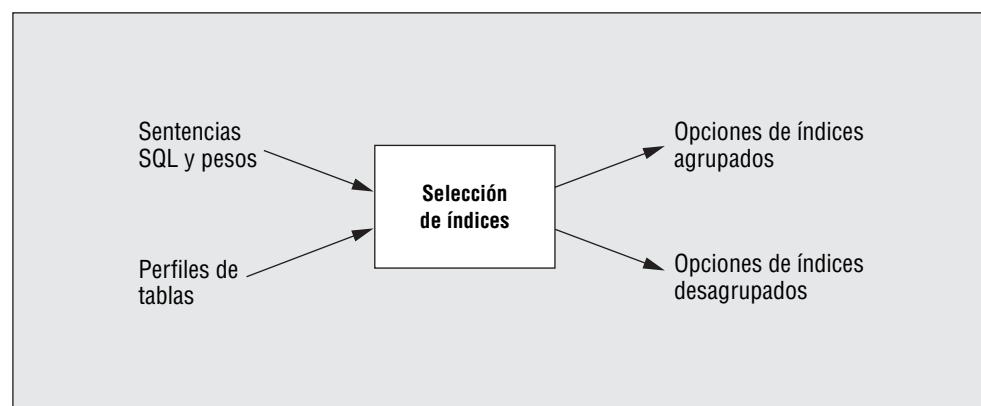
físicos que contienen las filas no serán consultados más de una vez si el índice se consulta de forma secuencial. La figura 8.20 muestra el conjunto de secuencias de un índice B+Tree que apunta a las filas asociadas dentro de los registros físicos. Observe que para un nodo determinado del conjunto de secuencias, la mayoría de las filas asociadas están agrupadas dentro del mismo registro físico. La forma más sencilla de hacer un índice agrupado es ordenando los datos de las filas mediante la columna índice.

En contraste, un índice desagrupado no tiene esta propiedad de cercanía. En un índice desagrupado el orden de las filas no está relacionado con el orden del índice. La figura 8.21 muestra que el mismo registro físico puede ser consultado de forma repetitiva cuando se usa un conjunto de secuencias. Los apuntadores de los nodos del conjunto de secuencias hacia las filas se cruzan muchas veces, indicando que el orden del índice es distinto al orden de las filas.

**FIGURA 8.21** Ejemplo de índice desagrupado



**FIGURA 8.22**  
Entradas y salidas  
de la selección de un  
índice



**problema de la selección de índices** para cada tabla, seleccione como máximo un índice agrupado y cero o más índices desagrupados.

La selección de índices incluye opciones acerca de índices agrupados y desagrupados, como se muestra en la figura 8.22. Generalmente se asume que cada tabla se almacena en un archivo. Las sentencias SQL señalan el trabajo de base de datos que deben realizar las aplicaciones. Los pesos deben combinar la frecuencia de una sentencia con su importancia. Los perfiles de tablas se deben especificar con el mismo nivel de detalle como lo requiera la optimización de la consulta.

Generalmente, el problema de selección de índices está restringido a índices Btree y a archivos separados para cada tabla. Las referencias al final del capítulo proporcionan detalles sobre el uso de otros tipos de índices (tales como índices hash) y la ubicación de datos de varias tablas en el mismo archivo. Sin embargo, estas ampliaciones hacen que el problema sea más difícil si no se agregan muchas mejoras al desempeño. Estas ampliaciones son útiles únicamente en situaciones especiales.

### 8.5.2 Equilibrios y dificultades

La mejor selección de índices balancea una recuperación rápida con actualizaciones más lentas. Un índice desagrupado puede mejorar las recuperaciones al proporcionar un acceso rápido a los registros seleccionados. En el ejemplo 8.2, un índice desagrupado en las columnas *OffYear*, *OffTerm* o *CourseNo* puede ser útil si relativamente pocas filas satisfacen la condición de asociación de la consulta. Generalmente, menos de 5 por ciento de las filas deben satisfacer una condición para que el índice desagrupado sea útil. Es poco probable que cualquiera de las condiciones del ejemplo 8.2 tenga una fracción tan pequeña de filas.

Para que los optimizadores soporten accesos a índices múltiples para la misma tabla, los índices desagrupados pueden ser útiles incluso cuando un índice sencillo no proporcione por sí mismo suficiente selección de filas. Por ejemplo, el número de filas después de aplicar las condiciones sobre *CourseNo*, *OffYear* y *OffTerm* debe ser pequeño, tal vez de 20 a 30 filas. Si un optimizador puede estimar el número de filas, los índices de tres columnas se pueden combinar para acceder a las filas de *Offering*. Por ende, la habilidad de usar múltiples índices sobre la misma tabla aumenta la utilidad de los índices desagrupados.

Un índice desagrupado también puede ser útil en un *join* cuando una de las tablas del enlace tenga un número pequeño de filas en el resultado. Por ejemplo, si únicamente pocas filas de *Offering* cumplen las tres condiciones del ejemplo 8.2, un índice desagrupado sobre la columna *Faculty.FacSSN* puede ser útil cuando se enlacen las tablas *Faculty* y *Offering*.

#### EJEMPLO 8.2 (Oracle)

##### Join de las tablas Faculty y Offering

```
SELECT FacName, CourseNo, OfferNo
  FROM Offering, Faculty
 WHERE CourseNo LIKE 'IS%' AND OffYear = 2005
   AND OffTerm = 'FALL'
   AND Faculty.FacSSN = Offering.FacSSN
```

Un índice agrupado puede mejorar las recuperaciones en más situaciones que un índice desagrupado. Un índice agrupado es útil en las mismas situaciones que un índice desagrupado a excepción de que el número de filas resultantes puede ser mayor. Por ejemplo, un índice agrupado por las columnas *CourseNo*, *OffYear* u *OffTerm* puede ser útil si tal vez 20 por ciento de las filas satisfacen la condición asociada de la consulta.

Un índice agrupado también puede ser útil en los enlaces ya que evita la necesidad de ordenar. Por ejemplo, al usar índices agrupados sobre las columnas *Offering.FacSSN* y *Faculty.FacSSN*, las tablas *Offering* y *Faculty* se pueden enlazar mezclando las filas de cada tabla. La mezcla de filas generalmente es una forma más rápida para enlazar tablas cuando éstas no necesiten estar ordenadas (existen índices agrupados).

El costo de conservar índices como resultado de las sentencias INSERT, UPDATE y DELETE balancea las mejoras de recuperación. Las sentencias INSERT y DELETE afectan a todos los índices de una tabla. Por lo tanto, se prefiere que una tabla no tenga muchos índices cuando tiene operaciones frecuentes de inserción y eliminación. Las sentencias UPDATE afectan sólo

a las columnas enlistadas en la cláusula SET. Si las sentencias UPDATE hechas sobre una columna son frecuentes, se pierde el beneficio del índice.

Las alternativas de los índices agrupados son más sensibles al mantenimiento que las de los índices desagrupados. Los índices agrupados son más costosos de mantener que los índices desagrupados porque el archivo de datos debe cambiar de forma similar a como lo hace un archivo secuencial ordenado. Para los índices desagrupados, el archivo de datos se puede mantener como se hace con un archivo secuencial desordenado.

#### *Dificultades en la selección de índices*

Es difícil realizar la selección de índices por varias razones, que son las que se explican en la siguiente subsección. Si usted comprende las razones de la dificultad para la selección de índices, debe obtener habilidades con las herramientas asistidas por computadora para que le ayuden en el proceso de selección para bases de datos grandes. Los DBMS corporativos y algunos otros fabricantes proporcionan herramientas asistidas por computadora para ayudarle en la selección de índices.

- Es difícil especificar los pesos de las aplicaciones. Los juicios que combinan la frecuencia e importancia pueden hacer que el resultado sea subjetivo.
- En ocasiones se necesita la distribución de los valores de los parámetros. Muchas sentencias SQL usadas en los reportes y formularios usan valores en los parámetros. Si los valores de los parámetros varían desde ser muy selectivos hasta no serlo, la selección de índices es difícil.
- Se debe conocer el comportamiento del componente de optimización de consultas. Incluso si un índice parece ser útil para alguna consulta, el componente de optimización de consultas debe usarlo. Puede haber razones sutiles para que el componente de optimización de consultas no use un índice, en especial un índice desagrupado.
- El número de alternativas es grande. Incluso si los índices que están en las combinaciones de las columnas son ignorados, el número teórico de alternativas es exponencial con el número de columnas ( $2^{NC}$  en donde  $NC$  es el número de columnas). Aunque muchas de estas alternativas se pueden eliminar fácilmente, el número de alternativas prácticas todavía es muy grande.
- Las alternativas de los índices se pueden interrelacionar. Las interrelaciones pueden ser sutiles, en especial cuando la selección de índices puede mejorar el desempeño de los *join*.

Una herramienta para seleccionar índices puede ayudarle con los tres últimos problemas. Una herramienta adecuada debe usar el componente de optimización de consultas para deducir las estimaciones de costo para cada consulta bajo una determinada selección de índices. Sin embargo, una buena herramienta no puede ayudarle a mitigar la dificultad para especificar los perfiles de las aplicaciones y las distribuciones de los valores de los parámetros. Se pueden proporcionar otras herramientas para especificar y capturar los perfiles de la aplicación.

#### **8.5.3 Reglas de selección**

A pesar de las dificultades previamente comentadas, generalmente podrá evitar la selección de malos índices siguiendo algunas reglas simples. También puede usar las reglas como un punto de partida de un proceso de selección más cuidadoso.

**Regla 1:** Una llave primaria es un buen candidato para un índice agrupado.

**Regla 2:** Para respaldar los *join*, considere los índices sobre las llaves foráneas. Un índice desagrupado sobre una llave foránea es una buena idea cuando existen consultas importantes con condiciones altamente selectivas hechas sobre una tabla relacionada. Un índice agrupado es una buena opción cuando la mayoría de los enlaces usan una tabla madre con un índice agrupado sobre su llave primaria, y las consultas no tienen condiciones altamente selectivas sobre la tabla madre.

**Regla 3:** Una columna con muchos valores puede ser una buena opción para un índice desagrupado cuando se usan en condiciones de igualdad. El término *muchos valores* significa que la columna es casi única.

- Regla 4:** Una columna que se usa en un rango de condiciones altamente selectivas es buena candidata para convertirse en un índice desagrupado.
- Regla 5:** Una combinación de columnas usada en forma conjunta con las condiciones de una consulta puede ser candidata para convertirse en índices desagrupados cuando las condiciones del enlace regresen pocas filas, el optimizador del DBMS soporte el acceso a varios índices y las columnas sean estables. Los índices individuales deben crearse para cada columna.
- Regla 6:** Una columna que se actualiza frecuentemente no es un buen candidato para un índice.
- Regla 7:** Las tablas volátiles (con muchas inserciones y eliminaciones) no deben tener muchos índices.
- Regla 8:** Las columnas estables con pocos valores son candidatas a convertirse en índices de tipo bitmap cuando las columnas se encuentren dentro de las condiciones WHERE.
- Regla 9:** Evite los índices con combinaciones de columnas. La mayoría de los componentes de optimización pueden usar varios índices sobre la misma tabla. Un índice que se haga sobre una combinación de columnas no es tan flexible como varios índices que se hagan para las columnas individuales de la tabla.

#### Aplicación de las reglas de selección

Apliquemos estas reglas a las tablas *Student*, *Enrollment* y *Offering* de la base de datos de la universidad. La tabla 8.10 lista las sentencias SQL y las frecuencias para esas tablas. Los nombres

**TABLA 8.10** Sentencias SQL y frecuencias para diversas tablas de la base de datos de la universidad

Sentencia SQL	Frecuencia	Comentarios
1. <i>INSERT INTO Student . . .</i>	7 500/año	Inicio del año
2. <i>INSERT INTO Enrollment . . .</i>	120 000/periodo	Durante el registro
3. <i>INSERT INTO Offering . . .</i>	1 000/año	Antes del tiempo límite del calendario
4. <i>DELETE Student WHERE StdSSN = \$X</i>	8 000/año	Después de la separación
5. <i>DELETE Offering WHERE OfferNo = \$X</i>	1 000/año	Al final del año
6. <i>DELETE Enrollment WHERE OfferNo = \$X AND StdSSN = \$Y</i>	64 000/año	Al final del año
7. <i>SELECT * FROM Student WHERE StdGPA &gt; \$X AND StdMajor = \$Y</i>	1 200/año	\$X es generalmente muy grande o pequeño
8. <i>SELECT * FROM Student WHERE StdSSN = \$X</i>	30 000/periodo	
9. <i>SELECT * FROM Offering WHERE OffTerm = \$X AND OffYear = \$Y AND CourseNo LIKE \$Z%</i>	60 000/periodo	Pocas filas en el resultado
10. <i>SELECT * FROM Offering, Enrollment WHERE StdSSN = \$X AND OffTerm = \$Y AND OffYear = \$Z AND Offer.OfferNo = Enrollment.OfferNo</i>	30 000/periodo	Pocas filas en el resultado
11. <i>UPDATE Student SET StdGPA = \$X WHERE StdSSN = \$Y</i>	30 000/periodo	Actualizados al final del formulario de reportes
12. <i>UPDATE Enrollment SET EnrGrade = \$X WHERE StdSSN = \$Y AND OfferNo = \$Z</i>	120 000/periodo	Parte del formulario de reportes de grados
13. <i>UPDATE OfferNo SET FacSSN = \$X WHERE OfferNo = \$Y</i>	500/año	
14. <i>SELECT FacSSN, FacFirstName, FacLastName FROM Faculty WHERE FacRank = \$X AND FacDept = \$Y</i>	1 000/periodo	La mayoría ocurre durante el registro
15. <i>SELECT * FROM Student, Enrollment, Offering WHERE Offer.OfferNo = \$X AND Student.StdSSN = Enrollment.StdSSN AND Offer.OfferNo = Enrollment.OfferNo</i>	4 000/año	La mayoría ocurre al inicio del semestre

**TABLA 8.11**  
Perfiles de las tablas

Tabla	Número de filas	Columna (número de valores únicos)
Estudiante	30 000	StdSSN (PK), StdLastName (29 000), StdAddress (20 000), StdCity (500), StdZip (1 000), StdState (50), StdMajor (100), StdGPA (400)
Inscripción	300 000	StdSSN (30 000), OfferNo (2 000), EnrGrade (400)
Oferta	10 000	OfferNo (PK), CourseNo (900), OffTime (20), OffLocation (500), FacSSN (1 500), OffTerm (4), OffYear (10), OffDays (10)
Curso	1000	CourseNo (PK), CrsDesc (1 000), CrsUnits (6)
Catedrático	2 000	FacSSN (PK), FacLastName (1 900), FacAddress (1 950), FacCity (50), FacZip (200), FacState (3), FacHireDate (300), FacSalary (1 500), FacRank (10), FacDept (100)

**TABLA 8.12**  
Selección de índices para las tablas de la base de datos de la universidad

Columna	Tipo de índice	Regla
Student.StdSSN	Agrupado	1
Student.StdGPA	Desagrupado	4
Offering.OfferNo	Agrupado	1
Enrollment.OfferNo	Agrupado	2
Faculty.FacRank	Bitmap	8
Faculty.Dept	Bitmap	8
Offering.OffTerm	Bitmap	8
Offering.OffYear	Bitmap	8

que comienzan con \$ representan los parámetros introducidos por un usuario. Las frecuencias asumen que existe una población de 30 000 estudiantes, los cuales se inscriben a un promedio de cuatro cursos por periodo. Después de que un estudiante se gradúa o deja la universidad, se almacenan las filas de *Student* y *Enrollment*. La tabla 8.11 lista los resúmenes de los perfiles de las tablas. Se puede codificar mayor detalle sobre la distribución de las columnas y las relaciones mediante histogramas.

La tabla 8.12 lista las opciones de índices de acuerdo con las reglas de selección de índices. Sólo se recomiendan pocos índices dada la frecuencia de las sentencias de mantenimiento y la ausencia de condiciones altamente selectivas sobre columnas que no son la llave primaria. En las consultas 9 y 10, aunque las condiciones individuales sobre *OffTerm* y *OffYear* no son altamente selectivas, la condición de enlace puede ser razonablemente selectiva como para recomendar índices bitmap, en especial, en la consulta 9, con la condición adicional de *CourseNo*. Existe un índice sobre *StdGPA* porque los valores de los parámetros deben ser muy altos o bajos, y con esto proporcionan una alta selectividad con pocas filas en el resultado. Puede ser necesario un estudio más detallado sobre el índice *StdGPA* ya que tiene una cantidad considerable de actualizaciones. Aunque no esté sugerido por las sentencias SQL, las columnas *STDLastName* y *FacLastName* también pueden ser buenas opciones de índices, ya que casi son únicas (con pocos duplicados) y razonablemente estables. Cuando existan sentencias SQL adicionales que usen estas columnas en sus condiciones, se deben considerar los índices desagrupados.

Aunque SQL:2003 no soporta las sentencias de índices, la mayoría de los DBMS lo hacen. En el ejemplo 8.3, la palabra que se muestra enseguida de la palabra clave INDEX es el nombre del índice. La sentencia CREATE INDEX también se puede usar para crear un índice sobre una combinación de columnas, listando las distintas columnas entre paréntesis. La sentencia CREATE INDEX, de Oracle, no se puede usar para crear un índice agrupado. Para crearlo, Oracle proporciona la cláusula ORGANIZATION INDEX como parte de la sentencia CREATE TABLE.

<b>EJEMPLO 8.3 (Oracle)</b>	<b>Sentencias CREATE INDEX</b>
	CREATE UNIQUE INDEX StdSSNIndex ON Student (StdSSN)
	CREATE INDEX StdGPAIndex ON Student (StdGPA)
	CREATE UNIQUE INDEX OfferNoIndex ON Offering (OfferNo)
	CREATE INDEX EnrollOfferNoIndex ON Enrollment (OfferNo)
	CREATE BITMAP INDEX OffYearIndex ON Offering (OffYear)
	CREATE BITMAP INDEX OffTermIndex ON Offering (OffTerm)
	CREATE BITMAP INDEX FacRankIndex ON Faculty (FacRank)
	CREATE BITMAP INDEX FacDeptIndex ON Faculty (FacDept)

## 8.6 Opciones adicionales del diseño físico de base de datos

Aunque la selección de índices es la decisión más importante del diseño físico de las bases de datos, existen otras decisiones que pueden mejorar el desempeño de manera significativa. Esta sección describe dos decisiones que pueden mejorar el desempeño en determinadas situaciones: la desnormalización y el formateo de registros. Más adelante, esta sección presenta una alternativa muy popular: el procesamiento en paralelo para mejorar el desempeño de las base de datos. Finalmente, se describen distintas maneras de mejorar el desempeño en relación con tipos específicos de procesamiento.

### 8.6.1 Desnormalización

#### diseños normalizados

- Tienen un mejor desempeño para las actualizaciones.
- Requieren menos código para obligar a que se cumplan las restricciones de integridad.
- Soportan más índices para mejorar el desempeño de las consultas.

La desnormalización combina tablas para que sean más fáciles de consultar. Después de combinar las tablas, la nueva tabla puede violar alguna de las formas normales como el BCNF. Aunque algunas de las técnicas de desnormalización no conducen a violaciones de alguna forma normal, hacen que el diseño sea más fácil de consultar y más difícil de actualizar. La desnormalización siempre debe hacerse con mucho cuidado ya que un diseño normalizado tiene importantes ventajas. El capítulo 7 describió una situación de la desnormalización: ignorar una dependencia funcional si no conduce a anomalías significativas de las modificaciones. Esta sección describe situaciones adicionales con las que se podría justificar la desnormalización.

#### Grupos de repetición

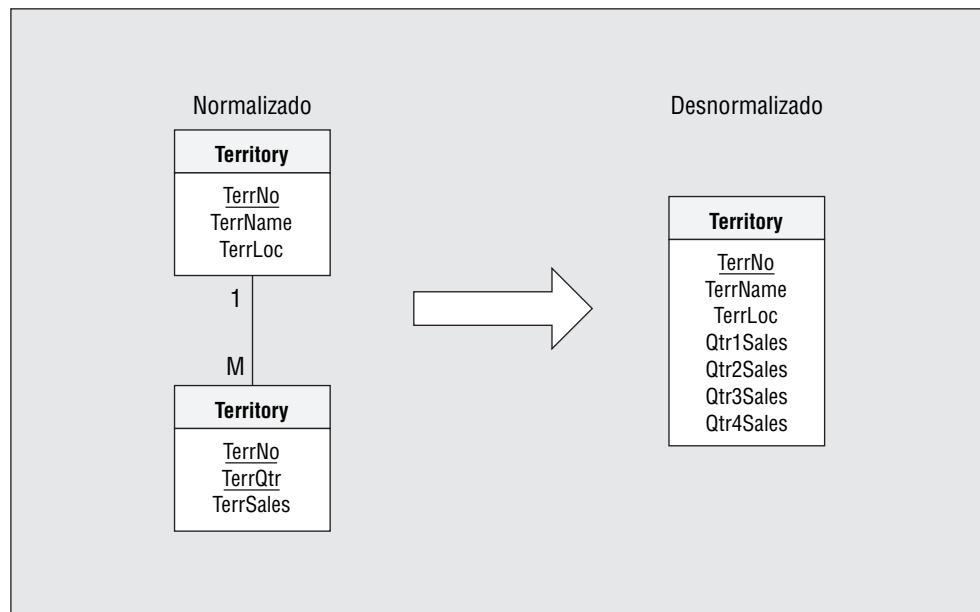
Un grupo de repetición es un conjunto de valores asociados como la historia de las ventas, los elementos de una orden, o el historial de pagos. Las reglas de normalización obligan a que los grupos de repetición se almacenen en una tabla hija separada de su tabla madre asociada. Por ejemplo, los elementos de una orden se almacenan en una tabla de elementos de órdenes, separados de la tabla relacionada de órdenes. La desnormalización puede ser una alternativa factible si siempre se accede a un grupo repetido mediante su tabla madre asociada.

La figura 8.23 muestra un ejemplo de desnormalización de los datos cuatrimestrales de ventas. Aunque el diseño desnormalizado no viola la forma normal BCNF, es menos flexible para las actualizaciones que el diseño normalizado. Un diseño de este tipo soporta un número ilimitado de ventas cuatrimestrales, a diferencia de los únicos cuatro cuatrimestres de resultados de ventas del diseño desnormalizado. Sin embargo, el diseño desnormalizado no requiere un *join* para combinar los datos de territorio y ventas.

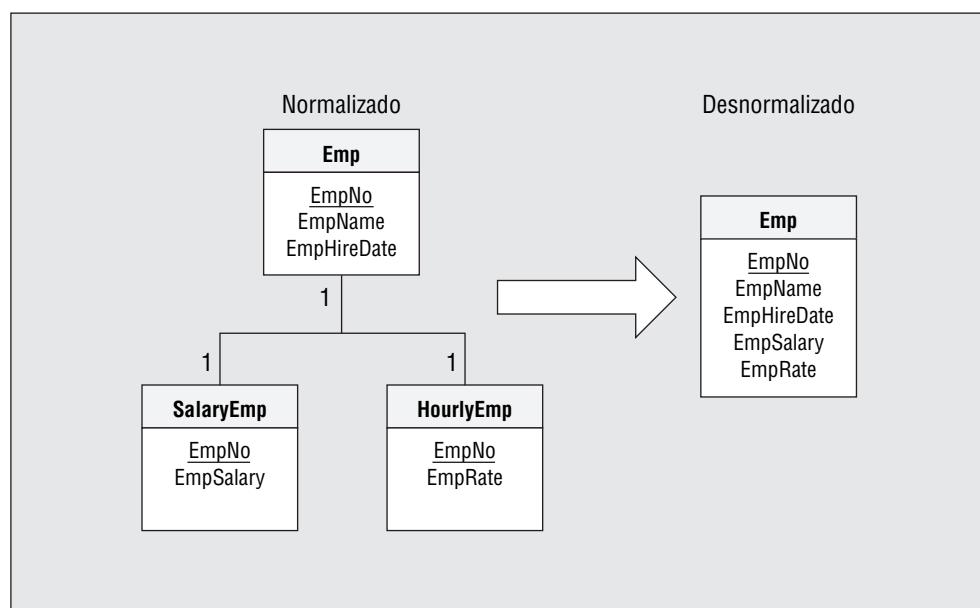
#### Jerarquías de generalización

Si se sigue la regla de conversión de jerarquías de generalización del capítulo 6, se pueden obtener muchas tablas. Si las consultas necesitan combinar regularmente estas tablas separadas, es factible almacenarlas como una sola tabla. La figura 8.24 muestra la desnormalización de las tablas *Emp*, *HourlyEmp* y *SalaryEmp*. Las tablas tienen relaciones 1-1, ya que representan una jerarquía de generalización. Aunque el diseño desnormalizado no viola la BCNF, la tabla combinada puede

**FIGURA 8.23**  
Desnormalización de un grupo repetido



**FIGURA 8.24**  
Desnormalización de una jerarquía de generalización



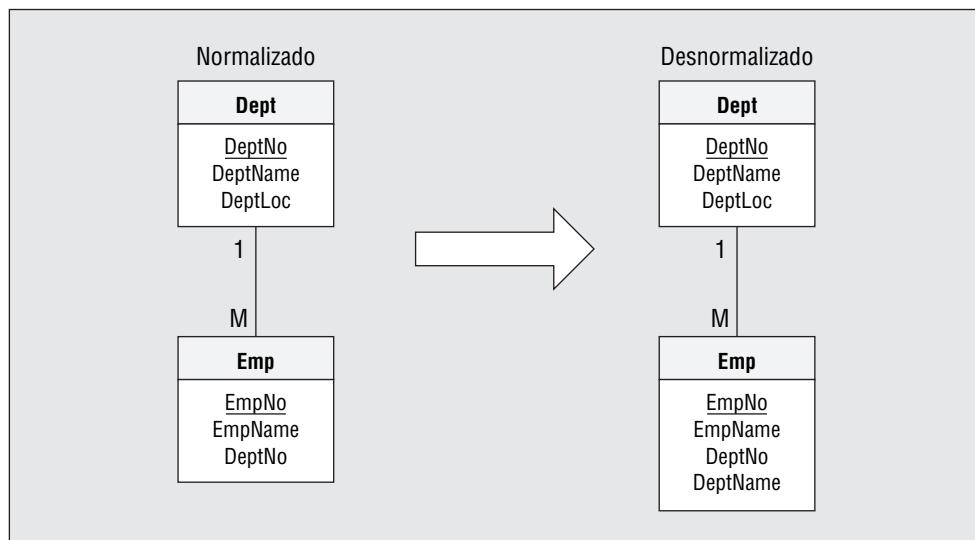
desperdiciar mucho espacio debido a los valores nulos; sin embargo, el diseño desnormalizado evita el uso del operador de *join* externos (*outer join*) para combinar las tablas.

#### Códigos y significados

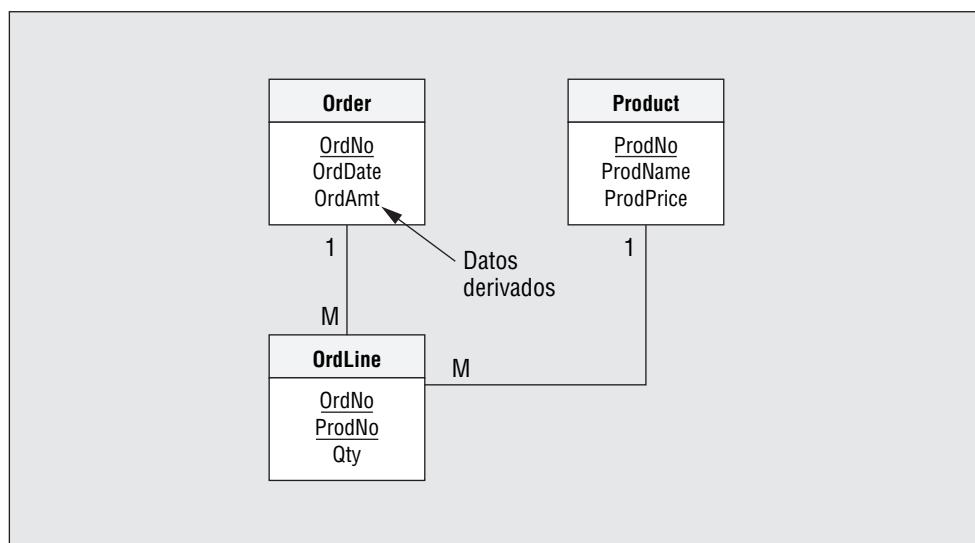
Las reglas de normalización requieren que las llaves foráneas se almacenen en forma aislada para representar las relaciones 1-M. Si una llave foránea representa un código, generalmente el usuario necesita un nombre asociado o descripción, además del valor de la llave foránea. El almacenamiento de la columna del nombre o de la descripción junto con el código viola la BCNF, pero elimina algunas operaciones de *join*. La desnormalización puede ser una opción razonable si la columna del nombre o de la descripción no cambia de forma constante. La figura 8.25 muestra la desnormalización para las tablas *Dept* y *Emp*. En el diseño desnormalizado la columna *DeptName* se agregó a la tabla *Emp*.

**FIGURA 8.25**

**Desnormalización para combinar las columnas de códigos y significados**

**FIGURA 8.26**

**Almacenamiento de los datos derivados para mejorar el desempeño de las consultas**



### 8.6.2 Formateo de registros

Las decisiones sobre el formateo de registros incluyen la compresión y datos derivados. La compresión se está convirtiendo en un elemento importante con mayor énfasis en el almacenamiento de tipos de datos complejos, tales como audio, video e imágenes. La compresión tiene sus ventajas y desventajas con respecto al esfuerzo del procesamiento de entradas-salidas. La compresión reduce el número de registros físicos transferidos, pero puede requerir de un esfuerzo de procesamiento considerable para comprimir y descomprimir los datos.

Las decisiones sobre datos derivados involucran ventajas y desventajas entre las operaciones de consultas y actualizaciones. Para efectos de consultas, el almacenamiento de datos derivados reduce la necesidad de recuperar datos requeridos para calcular los datos derivados. Sin embargo, las actualizaciones para los datos utilizados en el cálculo requieren de actualizaciones adicionales de los datos derivados. Puede ser razonable almacenar datos derivados para reducir las operaciones de *join*. La figura 8.26 muestra los datos derivados de la tabla *Order*. Si la cantidad total de una orden se consulta con cierta frecuencia, puede ser razonable almacenar la columna derivada *OrdAmt*. El cálculo del monto de una orden requiere de un resumen

o de cálculos adicionales de las filas relacionadas *OrdLine* y *Product*, con el fin de obtener las columnas *Qty* y *ProdPrice*. El almacenamiento de la columna *OrdAmt* evita que se hagan dos operaciones de enlace.

### 8.6.3 Procesamiento paralelo

El desempeño se puede mejorar de forma significativa al efectuar operaciones de recuperación y modificación a través del procesamiento paralelo. Las recuperaciones que involucren muchos registros se pueden mejorar al leer los registros físicos en paralelo. Por ejemplo, un reporte que agrupa la actividad de las ventas diarias logra leer miles de registros de distintas tablas. Además, el desempeño suele mejorarse de forma significativa para las aplicaciones que trabajan por lotes con muchas operaciones de escritura y de lectura/escritura de enormes registros físicos, tales como las imágenes.

Como respuesta a las posibles mejoras en el desempeño, muchos DBMS proporcionan capacidades de procesamiento en paralelo. El capítulo 17 describe las arquitecturas para el procesamiento en paralelo de bases de datos. Esta presentación se limita a una parte importante de cualquier arquitectura de procesamiento en paralelo de bases de datos, los arreglos redundantes de discos independientes (RAID).<sup>2</sup> El controlador RAID (figura 8.27) permite que un arreglo de discos se muestre al DBMS como un disco único muy grande. Para obtener un alto desempeño, el controlador RAID puede controlar hasta 90 discos. Debido a este controlador, el almacenamiento RAID no requiere cambios que deban tomarse en cuenta para el procesamiento en paralelo al hacer la evaluación de un plan de acceso.

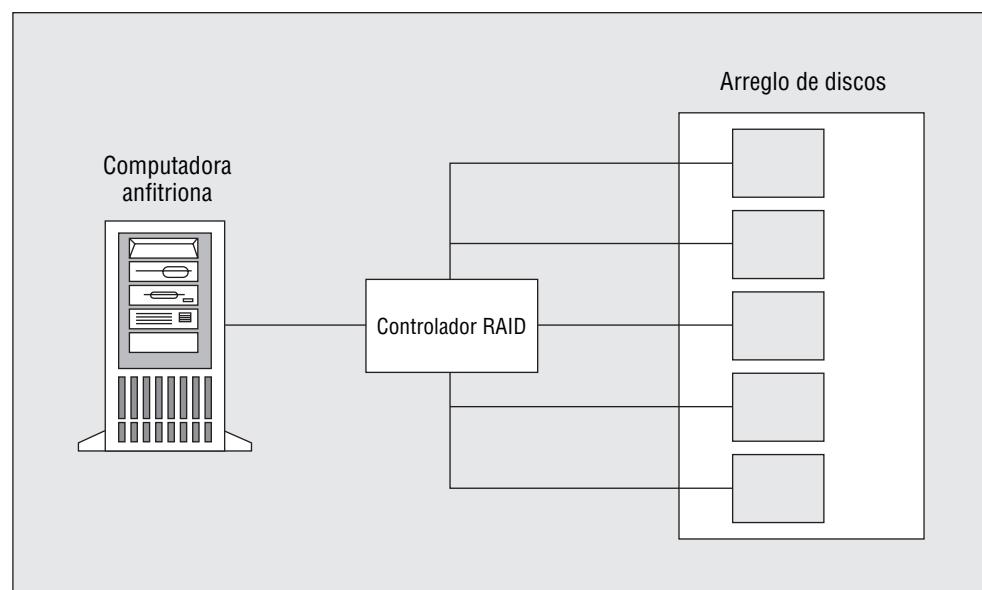
La distribución es un concepto importante del almacenamiento RAID. La distribución incluye la colocación de los registros físicos en distintos discos. Una distribución es un conjunto de registros físicos que pueden leerse o escribirse en paralelo. Normalmente, una distribución contiene un conjunto de registros físicos adyacentes. La figura 8.28 muestra un arreglo de cuatro discos que permiten leer o escribir cuatro registros físicos en paralelo.

Para utilizar el almacenamiento RAID han surgido varias arquitecturas. Las arquitecturas, conocidas como RAID-X, soportan el procesamiento en paralelo con diferentes elementos de desempeño y confiabilidad. La confianza es un elemento importante ya que el tiempo entre las fallas (una medida de la confiabilidad de los discos) disminuye conforme aumenta el número

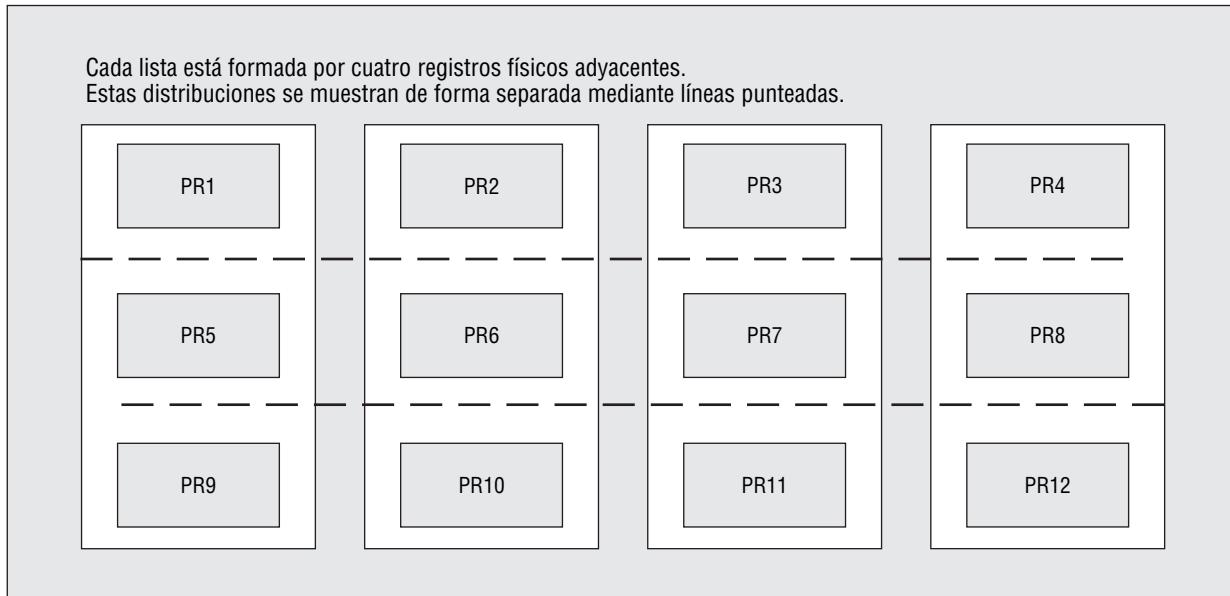
#### RAID

una colección de discos (un arreglo de discos) que operan como un solo disco. El almacenamiento RAID soporta operaciones de lectura y escritura en paralelo con una alta confiabilidad.

**FIGURA 8.27**  
Componentes de un sistema de almacenamiento RAID



<sup>2</sup> RAID originalmente era el acrónimo de Arreglos Redundantes de Discos no Costosos (*Redundant Arrays of Inexpensive Disks*). Debido a que los precios de los discos han bajado de forma dramática desde la invención de la idea RAID (1988), las palabras *no Costosos* han sido reemplazadas por *independiente*.

**FIGURA 8.28** Lista de sistemas de almacenamiento RAID

de discos. Para combatir las preocupaciones de la confiabilidad, las arquitecturas RAID incorporan redundancia con el uso de discos en espejo, códigos de corrección de errores y discos de repuesto. Para la mayoría de estos propósitos dominan dos arquitecturas RAID, aunque existen muchas variaciones de estas arquitecturas básicas.

- **RAID-1:** incluye un espejo completo o arreglo redundante de discos para mejorar la confiabilidad. Cada registro físico se escribe en los dos arreglos de discos en paralelo. Las operaciones de lectura de consultas separadas pueden acceder al arreglo de discos en paralelo para mejorar el desempeño entre las consultas. RAID-1 incluye la mayor sobrecarga de almacenamiento comparada con otras arquitecturas RAID.
- **RAID-5:** utiliza tanto las páginas de datos como de corrección de errores (conocidas como páginas de paridad) para mejorar la confiabilidad. Las operaciones de lectura se pueden llevar a cabo en paralelo en las distribuciones. Las operaciones de escritura involucran una página de datos y una página de corrección de errores en otro disco. Para reducir la contención de los discos, las páginas de corrección de errores se localizan de manera aleatoria entre los discos. RAID-5 usa el espacio de almacenamiento de forma más eficiente que RAID-1, pero puede llevar a tiempos de escritura más lentos debido a las páginas de corrección de errores. Por lo general, se prefiere RAID-1 para las partes altamente volátiles de una base de datos.

Para aumentar la capacidad más allá de RAID y eliminar la dependencia de los dispositivos de almacenamiento propios del servidor, se han desarrollado las redes de áreas de almacenamiento (*Storage Area Networks* o SANs). Una SAN es una red especializada de alta velocidad que conecta los dispositivos de almacenamiento y los servidores. El objetivo de la tecnología SAN es integrar de forma sencilla distintos tipos de subsistemas de almacenamiento en un solo sistema y eliminar el potencial cuello de botella de un único servidor que controle los dispositivos de almacenamiento. Muchas organizaciones grandes usan las SAN para integrar sistemas de almacenamiento de bases de datos operacionales, *data warehouses*, almacenamiento histórico de documentos y sistemas de archivos tradicionales.

#### 8.6.4 Otras formas de mejorar el desempeño

Existen otras formas de mejorar el desempeño de una base de datos que están relacionadas con un tipo específico de procesamiento. Para el procesamiento transaccional (capítulo 15), usted puede agregar capacidad de cómputo (más procesadores y más rápidos, memoria y disco duro) y

evaluar las ventajas y desventajas en el diseño transaccional. Para los *data warehouses* (capítulo 16) puede agregar capacidad de cómputo y diseñar tablas nuevas con datos derivados. Para el procesamiento de bases de datos distribuidas (capítulo 17) puede colocar el procesamiento y los datos en varias computadoras. Los datos se pueden colocar dividiendo una tabla de forma vertical (subconjunto de columnas) y horizontal (subconjunto de filas) para ubicar los datos cerca de donde se usen. Estas opciones de diseño se describen en los capítulos respectivos de la parte 7.

Además de ajustar el desempeño para requerimientos específicos de procesamiento, usted también puede mejorar el desempeño utilizando opciones específicas de un DBMS. La mayoría de los DBMS proporcionan guías y herramientas para monitorear y controlar la fragmentación. Además, la mayoría de los DBMS tienen opciones para las estructuras de archivos que pueden ayudarle a mejorar el desempeño. Usted debe estudiar de manera cuidadosa su DBMS específico para comprender estas opciones. Puede llevarle varios años de experiencia y educación especializada comprender las opciones específicas de un DBMS en particular. Sin embargo, un mayor salario y la demanda de su conocimiento pueden hacer que valga la pena.

## Reflexión final

Este capítulo describió la naturaleza del proceso de diseño físico de bases de datos y los detalles de las entradas, ambiente y decisiones de diseño. El diseño físico de bases de datos incluye detalles muy cercanos al sistema operativo, tales como el movimiento de registros físicos. El objetivo del diseño físico de bases de datos es minimizar el uso de ciertos recursos de cómputo (accesos a registros físicos y esfuerzo de procesamiento) sin comprometer el significado de la base de datos. El diseño físico de bases de datos es un proceso difícil, ya que las entradas no son fáciles de especificar, el entorno es complejo y el número de opciones puede ser abrumador.

Para mejorar su conocimiento en la realización del diseño físico de bases de datos, este capítulo describió los detalles acerca de las entradas y del entorno del diseño físico de bases de datos. Este capítulo describió los perfiles de las tablas y de las aplicaciones como entradas que deben especificarse con el detalle suficiente para obtener un diseño eficiente. El entorno está formado por las estructuras de archivos y el componente de optimización de consultas del DBMS. Para las estructuras de archivos, este capítulo describió las características de las estructuras secuencial, hash, Btree y bitmap utilizadas por muchos DBMS. Para la optimización de consultas, este capítulo describió las tareas de la optimización de consultas y consejos para generar mejores resultados con la optimización.

Después de establecer los antecedentes del proceso de diseño físico de bases de datos, las entradas y el entorno, este capítulo describió las decisiones relacionadas con la selección de índices, desnormalización y formateo de registros. En lo que respecta a la selección de índices, este capítulo describió las ventajas y desventajas entre las aplicaciones de recuperación y actualización, y presentó las reglas para la selección de índices. También presentó varias situaciones en las que pueden ser útiles la desnormalización y el formateo de datos.

El presente capítulo concluye el proceso de desarrollo de una base de datos. Después de haber terminado estos pasos, usted deberá tener un diseño de tablas eficiente que represente las necesidades de una organización. Para completar su comprensión del proceso de desarrollo de bases de datos, el capítulo 13 proporciona un caso de estudio detallado en el cual puede aplicar las ideas de las partes previas de este libro.

## Revisión de conceptos

- Relación entre los registros físicos y lógicos.
- Objetivo del diseño físico de bases de datos.
- Dificultades del diseño físico de bases de datos.
- Nivel de detalle de los perfiles de tablas y aplicaciones.
- Histogramas de la misma altura para especificar las distribuciones de las columnas.
- Características de las estructuras de archivos secuencial, hash y Btree.
- Posibles significados de la letra *B* en el nombre Btree: balanceado, con arbustos, orientado a bloques.

- Índices bitmap para las columnas estables con pocos valores.
- Índices de *join* bitmap para operaciones frecuentes de enlace usando condiciones en columnas estables de no enlace.
- Tareas de la traducción del lenguaje de datos.
- El uso de las fórmulas de costos y de los perfiles de tablas para evaluar los planes de acceso.
- La importancia de los perfiles de las tablas con el detalle suficiente para poder evaluar los planes de acceso.
- Prácticas de codificación para evitar ejecutar consultas mal planteadas.
- La diferencia entre índices agrupados y desagrupados.
- Ventajas y desventajas al seleccionar índices.
- Las reglas de selección de índices para evitar las opciones de índices mal planteados.
- Desnormalización para mejorar el desempeño de enlace.
- Formateo de registros para reducir los accesos a registros físicos y mejorar el desempeño de las consultas.
- Almacenamiento RAID para proporcionar procesamiento en paralelo para recuperaciones y actualizaciones.
- Arquitecturas RAID para proporcionar procesamiento en paralelo con alta confiabilidad.
- Redes de áreas de almacenamiento (SANs) para integrar subsistemas de almacenamiento y eliminar la dependencia de un servidor que controle los dispositivos de almacenamiento.

## Preguntas

1. ¿Cuál es la diferencia entre un acceso a registros físicos y un acceso a registros lógicos?
2. ¿Por qué es difícil saber cuándo un acceso de registros lógicos resulta en un acceso a registros físicos?
3. ¿Cuál es el objetivo del diseño físico de una base de datos?
4. ¿Qué recursos de cómputo son restricciones en vez de formar parte del objetivo de un diseño físico de bases de datos?
5. ¿Cuáles son los contenidos de los perfiles de una tabla?
6. ¿Cuáles son los contenidos de los perfiles de una aplicación?
7. Describa dos formas de especificar las distribuciones de las columnas utilizadas en las tablas y los perfiles de las aplicaciones.
8. ¿Por qué la mayoría de los DBMS empresariales utilizan histogramas de la misma altura para representar las distribuciones de las columnas en lugar de emplear histogramas con el mismo intervalo?
9. ¿Qué es una estructura de archivo?
10. ¿Cuál es la diferencia entre una estructura de archivo primaria y una secundaria?
11. Describa el uso de los archivos secuenciales en una búsqueda secuencial, una búsqueda de rangos y una búsqueda por llaves.
12. ¿Cuál es el propósito de una función hash?
13. Describa los usos de los archivos hash en una búsqueda secuencial, una búsqueda de rangos y una búsqueda de llaves.
14. ¿Cuál es la diferencia entre una archivo hash estático y uno dinámico?
15. Defina los términos *balanceado*, *bushy* y *orientado a bloques* conforme se relacionan con los archivos Btree.
16. Explique brevemente el uso de las divisiones de nodos y los encadenamientos del mantenimiento de los archivos Btree.
17. ¿Qué significa decir que los Btrees tienen un costo de búsqueda logarítmica?
18. ¿Cuál es la diferencia entre un Btree y un B+Tree?
19. ¿Qué es un bitmap?
20. ¿Cómo utiliza un bitmap un DBMS?
21. ¿Cuáles son los componentes de un registro de índices de tipo bitmap?

22. ¿Cuál es la diferencia entre un índice de columna bitmap y un índice de enlace bitmap?
23. ¿Cuándo se deben usar los índices bitmap?
24. ¿Cuál es la diferencia entre una estructura de archivos primaria y una secundaria?
25. ¿Qué significa decir que un índice corresponde a una columna?
26. ¿Por qué deben utilizarse con poca frecuencia los índices compuestos?
27. ¿Qué sucede en la fase de transformación de consulta de la traducción del lenguaje de la base de datos?
28. ¿Qué es un plan de acceso?
29. ¿Qué es un barrido de índices múltiples?
30. ¿Cómo se evalúan los planes de acceso en una optimización de consulta?
31. ¿Por qué algunas veces la consideración uniforme del valor genera planes de acceso deficientes?
32. ¿Qué significa ligar una consulta?
33. ¿Qué algoritmo de *join* puede utilizarse para todas las operaciones de enlace?
34. ¿Para cuáles algoritmos de *join* debe elegir el componente de optimización a las tablas externas e internas?
35. ¿Qué algoritmo de *join* puede combinar más de dos tablas a la vez?
36. ¿Cuándo se considera al algoritmo *sort merge* como una buena opción para combinar tablas?
37. ¿Cuándo se considera a un algoritmo de *hash join* como una buena opción para combinar tablas?
38. ¿Qué es una sugerencia de optimizador? ¿Por qué las sugerencias deben usarse con cuidado?
39. Identifique una situación en la que no se deba usar una sugerencia del optimizador.
40. Identifique una situación en la que sea recomendable el uso de una sugerencia de optimizador.
41. ¿Cuál es la diferencia entre un índice agrupado y uno desagrupado?
42. ¿Cuándo resulta útil un índice desagrupado?
43. ¿Cuándo resulta útil un índice agrupado?
44. ¿Cuál es la relación entre la selección de un índice y la optimización de consultas?
45. ¿Cuáles son las ventajas y desventajas que se presentan en la selección de índices?
46. ¿Por qué es difícil una selección de índices?
47. ¿Cuándo se deben utilizar las reglas para la selección de índices?
48. ¿Por qué se debe tener cuidado con la desnормalización?
49. Identifique dos situaciones en las que la desnормalización resulta útil.
50. ¿Qué es un almacenamiento RAID?
51. ¿Para qué tipo de aplicaciones se puede mejorar el desempeño mediante el uso del almacenamiento RAID?
52. ¿Qué es una lista cuando se habla del almacenamiento RAID?
53. ¿Qué técnicas se utilizan en el almacenamiento RAID para mejorar la confiabilidad?
54. ¿Cuáles son las ventajas y desventajas de RAID-1 cuando se compara con RAID-5?
55. ¿Qué es una red de áreas de almacenamiento (SAN, por sus siglas en inglés)?
56. ¿Cuál es la relación entre el almacenamiento SAN y el almacenamiento RAID?
57. ¿Cuáles son las ventajas y desventajas en los datos que se derivan del almacenamiento?
58. ¿Qué ambientes de procesamiento también involucran decisiones del diseño físico de bases de datos?
59. ¿Cuáles son algunas preocupaciones específicas de DBMS para la mejora del desempeño?
60. ¿Qué es una conversión de tipo implícito? ¿Por qué puede causar un desempeño deficiente las conversiones de tipo implícito?
61. ¿Por qué los enlaces innecesarios ocasionan un desempeño deficiente de consultas?
62. ¿Por qué las condiciones de las filas de la cláusula HAVING deben desplazarse a la cláusula WHERE?

## Problemas

Además de los problemas que se presentan en esta sección, el estudio de caso del capítulo 13 proporciona una práctica adicional. Para complementar los ejemplos de este capítulo, el capítulo 13 ofrece un caso completo de diseño de una base de datos, incluyendo el diseño de una base de datos física.

1. Utilice los siguientes datos para realizar los cálculos indicados. Muestre las fórmulas que utilizó para realizar los cálculos.

Tamaño de la fila = 100 bytes

Número de filas = 100 000

Tamaño de la llave primaria = 6 bytes

Tamaño del registro físico = 4 096 bytes

Tamaño del apuntador = 4 bytes

Floor( $X$ ) es el número entero más grande que sea menor o igual a  $X$ .

Ceil( $X$ ) es el número entero más pequeño que sea mayor o igual a  $X$ .

- 1.1. Calcule el número de filas que pueden ajustarse a un registro físico. Considere que solamente pueden almacenarse filas completas (utilice la función Floor).
- 1.2. Calcule el número de registros físicos necesarios para un archivo secuencial. Considere que los registros físicos se llenan en su totalidad a excepción del último (utilice la función Ceil).
- 1.3. Si se utiliza un archivo secuencial desordenado, calcule el número de accesos al registro físico que se necesita en promedio para recuperar una fila con un valor de llave específico.
- 1.4. Si se utiliza un archivo secuencial ordenado, calcule el número de accesos al registro físico que se necesita en promedio para recuperar una fila con un valor específico de llave. Considere que la llave existe en el archivo.
- 1.5. Calcule el número promedio de accesos necesarios al registro físico para encontrar una llave que no exista en un archivo secuencial desordenado y en un archivo secuencial ordenado.
- 1.6. Calcule el número de registros físicos para un archivo hash estático. Considere que cada registro físico del archivo hash está a 70 por ciento de su capacidad.
- 1.7. Calcule el factor de número de ramas máximo en un nodo de Btree. Considere que cada registro en el Btree consiste de los pares <key value, pointer>.
- 1.8. Por medio del cálculo que realizó en el problema 1.7, calcule la altura máxima de un índice Btree.
- 1.9. Calcule el número máximo de accesos al registro físico necesarios para encontrar un nodo en Btree con un valor de llave específico.

2. Responda a las preguntas de optimización de consulta para la siguiente sentencia de SQL:

```
SELECT * FROM Customer
WHERE CustCity = 'DENVER' AND CustBalance > 5000
AND CustState = 'CO'
```

- 2.1. Muestre cuatro planes de acceso para esta consulta considerando que los índices no agrupados existen en las columnas *CustCity*, *CustBalance* y *CustState*. También hay un índice agrupado en la columna de la llave primaria *CustNo*.
- 2.2. Por medio de la consideración del valor uniforme, calcule la fracción de filas que satisfagan la condición en *CustBalance*. El balance más pequeño es 0 y el mayor es de \$10 000.
- 2.3. A partir del siguiente histograma, calcule la fracción de filas que satisfagan la condición en *CustBalance*.

**Histograma para *CustBalance***

Rango	Filas
0–100	1 000
101–250	950
251–500	1 050
501–1 000	1 030
1 001–2 000	975
2 001–4 500	1 035
4 501–	1 200

3. Responda las preguntas de optimización de consulta para la siguiente sentencia de SQL:

```

SELECT OrdNo, OrdDate, Vehicle.ModelNo
  FROM Customer, Order, Vehicle
 WHERE CustBalance > 5000
   AND Customer.CustNo = Vehicle.CustNo
   AND Vehicle.SerialNo = Order.SerialNo

```

- 3.1. Enliste las órdenes posibles para enlazar las tablas *Customer*, *Order* y *Vehicle*.
- 3.2. Elabore un plan de acceso para una de estas órdenes de enlace. Considere que los índices Btree existen solamente para las claves primarias, *Customer.CustNo*, *Order.OrdNo*. y *Vehicle.SerialNo*.
4. Para las siguientes tablas y sentencias de SQL, seleccione los índices que balancean los requerimientos de retiro y actualización. Justifique su elección para cada tabla por medio de las reglas que se comentaron en la sección 8.5.3.

```

Customer(CustNo, CustName, CustCity, CustState, CustZip, CustBal)
Order(OrdNo, OrdDate, CustNo)
    FOREIGN KEY CustNo REFERENCES Customer
OrdLine(OrdNo, ProdNo, OrdQty)
    FOREIGN KEY OrdNo REFERENCES Order
    FOREIGN KEY ProdNo REFERENCES Product
Product(ProdNo, ProdName, ProdColor, ProdPrice)

```

Sentencia de SQL	Frecuencia
1. <i>INSERT INTO Customer . . .</i>	100/día
2. <i>INSERT INTO Product . . .</i>	100/mes
3. <i>INSERT INTO Order . . .</i>	3 000/día
4. <i>INSERT INTO OrdLine . . .</i>	9 000/día
5. <i>DELETE Product WHERE ProdNo = \$X</i>	100/año
6. <i>DELETE Customer WHERE CustNo = \$X</i>	1 000/año
7. <i>SELECT * FROM Order, Customer WHERE OrdNo = \$X AND Order.CustNo = Customer.CustNo</i>	300/día
8. <i>SELECT * FROM OrdLine, Product WHERE OrdNo = \$X AND OrdLine.ProdNo = Product.ProdNo</i>	300/día
9. <i>SELECT * FROM Customer, Order, OrdLine, Product WHERE CustName = \$X AND OrdDate = \$Y AND Customer.CustNo = Order.CustNo AND Order.OrdNo = OrdLine.OrdNo AND Product.ProdNo = OrdLine.ProdNo</i>	500/día
10. <i>UPDATE OrdLine SET OrdQty = \$X WHERE OrdNo = \$Y</i>	300/día
11. <i>UPDATE Product SET ProdPrice = \$X WHERE ProdNo = \$Y</i>	300/mes

- 4.1. Para la tabla *Customer*, ¿qué columnas son buena elección para el índice agrupado? ¿Y para los índices desagrupados?
- 4.2. Para la tabla *Product*, ¿qué columnas son buena elección para el índice agrupado? ¿Y para los índices desagrupados?
- 4.3. Para la tabla *Order*, ¿qué columnas son buena elección para el índice agrupado? ¿Y para los índices desagrupados?
- 4.4. Para la tabla *OrdLine*, ¿qué columnas son buena elección para el índice agrupado? ¿Y para los índices desagrupados?
5. Los índices en las combinaciones de columnas no son tan útiles como los índices en columnas individuales. Considere un índice combinado en dos columnas, *CustState* y *CustCity*, en donde *CustState* es la ordenación primaria y *CustCity* la secundaria. ¿Para qué tipos de condiciones puede utilizarse el índice? ¿Para cuáles no resulta útil?
6. Para la consulta 9 del problema 4, enliste los posibles órdenes de *join* considerados por el componente de optimización de consultas.

7. Para la siguiente tabla de la base de datos de planeación financiera, identifique los posibles usos de la desnormalización y los datos derivados que ya aparecen en las tablas.

Las tablas presentan los diferentes activos financieros y las comercializaciones realizadas por los clientes. Una comercialización implica la compra o venta de determinada cantidad de un activo por parte de un cliente. Los activos incluyen acciones y bonos. La tabla *Holding* contiene la cantidad *neta* de cada activo que tiene un cliente. Por ejemplo, si un cliente ha comprado 10 000 acciones de IBM y vendido 4 000, la tabla *Holding* tiene una cantidad neta de 6 000. Una consulta frecuente consiste en hacer un listado de la evaluación más reciente de cada uno de los activos que tiene un cliente. La evaluación más reciente consiste en la cantidad neta del activo por el precio más reciente.

```
Customer(CustNo, CustName, CustAddress, CustCity, CustState, CustZip, CustPhone)
Asset(AssetNo, SecName, LastClose)
Stock(AssetNo, OutShares, IssShares)
Bond(AssetNo, BondRating, FacValue)
PriceHistory(AssetNo, PHistDate, PHistPrice)
    FOREIGN KEY AssetNo REFERENCES Asset
Holding(CustNo, AssetNo, NetQty)
    FOREIGN KEY CustNo REFERENCES Customer
    FOREIGN KEY AssetNo REFERENCES Asset
Trade(TradeNo, CustNo, AssetNo, TrdQty, TrdPrice, TrdDate, TrdType, TrdStatus)
    FOREIGN KEY CustNo REFERENCES Customer
    FOREIGN KEY AssetNo REFERENCES Asset
```

8. Reescriba la siguiente sentencia de SQL para mejorar su desempeño en la mayoría de los DBMS. Utilice las recomendaciones que aparecen en la sección 8.4.2 para reescribir la sentencia. La sentencia de Oracle SQL utiliza la base de datos de comercialización financiera que se mostró en el problema 7. El propósito de la sentencia consiste en hacer un listado del número y nombre de cliente así como la suma de la cantidad de sus comercializaciones de compra completadas hasta octubre de 2006. La cantidad de comercialización corresponde a la cantidad (número de acciones) por el precio por acción. Un cliente debe aparecer en el resultado si la suma de la cantidad de sus comercializaciones de compra completadas para octubre de 2006 excede por 25 por ciento a la suma de las cantidades de sus comercializaciones completadas para septiembre de 2006.

```
SELECT Customer.Custno, CustName,
       SUM(TrdQty * TrdPrice) AS SumTradeAmt
  FROM Customer, Trade
 WHERE Customer.CustNo = Trade.CustNo
   AND TrdDate BETWEEN '1-Oct-2006' AND '31-Oct-2006'
 GROUP BY Customer.CustNo, CustName
 HAVING TrdType = 'BUY' AND SUM(TrdQty * TrdPrice) >
    ( SELECT 1.25 * SUM(TrdQty * TrdPrice) FROM Trade
      WHERE TrdDate BETWEEN '1-Sep-2006' AND '30-Sep-2006'
        AND TrdType = 'BUY'
        AND Trade.CustNo = Customer.CustNo )
```

9. Reescriba la siguiente sentencia de SQL para mejorar su desempeño en la mayoría de los DBMS. Utilice las recomendaciones que aparecen en la sección 8.4.2 para reescribir la sentencia. La sentencia de Oracle SQL utiliza la base de datos de comercialización financiera que se mostró en el problema 7. Observe que la columna *CustNo* utiliza el tipo de datos enteros.

```
SELECT Customer.CustNo, CustName,
       TrdQty * TrdPrice, TrdDate, SecName
  FROM Customer, Trade, Asset
 WHERE Customer.CustNo = Trade.CustNo
   AND Trade.AssetNo = Asset.AssetNo
   AND TrdType = 'BUY'
   AND TrdDate BETWEEN '1-Oct-2006' AND '31-Oct-2006'
   AND Trade.CustNo = '10001'
```

10. Para las siguientes condiciones e índices, indique si el índice corresponde con la condición.
  - Índice en *TrdDate*: TrdDate BETWEEN '1-Oct-2006' AND '31-Oct-2006'
  - Índice en *CustPhone*: CustPhone LIKE '(303)%'
  - Índice en *TrdType* < > 'BUY'
  - Índice en la columna Bitmap en BondRating IN ('AAA', 'AA', 'A')
  - Índice en <*CustState*, *CustCity*, *CustZip*>:
    - *CustState* = 'CO' AND *CustCity* = 'Denver'
    - *CustState* IN ('CO', 'CA') AND *CustCity* LIKE '%er'
    - *CustState* IN ('CO', 'CA') AND *CustZip* LIKE '8%'
    - *CustState* = 'CO' AND *CustCity* IN ('Denver', 'Boulder') AND *CustZip* LIKE '8%'
  
11. Para el ejemplo de las tablas *Customer* y *Trade* que aparecen a continuación, construya índices de tipo bitmap tal como se indica.
  - Índice de tipo bitmap en la columna *Customer.CustState*.
  - Índice de tipo bitmap en *Customer.CustNo* hacia la tabla *Trade*.
  - Índice de tipo bitmap enlazado sobre *Customer.CustState* hacia la tabla *Trade*.

**Tabla de clientes**

RowID	CustNo	...	CustState
1	113344		CO
2	123789		CA
3	145789		UT
4	111245		NM
5	931034		CO
6	998245		CA
7	287341		UT
8	230432		CO
9	321588		CA
10	443356		CA
11	559211		UT
12	220688		NM

**Tabla de comercialización**

RowID	TradeNo	...	CustNo
1	1111		113344
2	1234		123789
3	1345		123789
4	1599		145789
5	1807		145789
6	1944		931034
7	2100		111245
8	2200		287341
9	2301		287341
10	2487		230432
11	2500		443356
12	2600		559211
13	2703		220688
14	2801		220688
15	2944		220688
16	3100		230432
17	3200		230432
18	3258		321588
19	3302		321588
20	3901		559211
21	4001		998245
22	4205		998245
23	4301		931034
24	4455		443356

12. Para las siguientes tablas y sentencias de SQL, elija los índices (agrupados y desagrupados) que balanceen los requerimientos de recuperación y actualización. Justifique su elección para cada tabla por medio de las reglas que se comentaron en la sección 8.5.3.

**Customer**(*CustNo*, *CustName*, *CustAddress*, *CustCity*, *CustState*, *CustZip*, *CustPhone*)

**Asset**(*AssetNo*, *AssetName*, *AssetType*)

**PriceHistory**(*AssetNo*, *PHistDate*, *PHistPrice*)

FOREIGN KEY *AssetNo* REFERENCES *Asset*

```

Holding(CustNo, AssetNo, NetQty)
    FOREIGN KEY CustNo REFERENCES Customer
    FOREIGN KEY AssetNo REFERENCES Asset
Trade(TradeNo, CustNo, AssetNo, TrdQty, TrdPrice, TrdDate, TrdType, TrdStatus)
    FOREIGN KEY CustNo REFERENCES Customer
    FOREIGN KEY AssetNo REFERENCES Asset

```

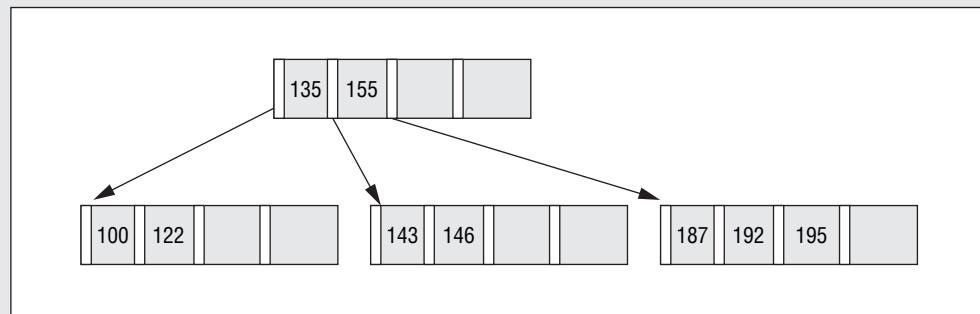
---

<b>Sentencia de SQL</b>	<b>Frecuencia</b>
1. <i>INSERT INTO Customer . . .</i>	100/día
2. <i>INSERT INTO Asset . . .</i>	100/trimestre
3. <i>INSERT INTO Trade . . .</i>	10 000/día
4. <i>INSERT INTO Holding . . .</i>	200/día
5. <i>INSERT INTO PriceHistory . . .</i>	5 000/día
5. <i>DELETE Asset WHERE AssetNo = \$X</i>	300/año
6. <i>DELETE Customer WHERE CustNo = \$X</i>	3 000/año
7. <i>SELECT * FROM Holding, Customer, Asset, PriceHistory WHERE CustNo = \$X AND Holding.CustNo = Customer.CustNo AND Holding.AssetNo = Asset.AssetNo AND Asset.AssetNo = PriceHistory.AssetNo AND PHistDate = \$Y</i>	15 000/mes
8. <i>SELECT * FROM Trade WHERE TradeNo = \$X</i>	1 000/día
9. <i>SELECT * FROM Customer, Trade, Asset WHERE Customer.CustNo = \$X AND TrdDate BETWEEN \$Y AND \$Z AND Customer.CustNo = Trade.CustNo AND Trade.AssetNo = Asset.AssetNo</i>	10 000/mes
10. <i>UPDATE Trade SET TrdStatus = \$X WHERE TradeNo = \$Y</i>	1 000/día
11. <i>UPDATE Holding SET NetQty = \$X WHERE CustNo = \$Y AND AssetNo = \$Z</i>	10 000/día
12. <i>SELECT * FROM Customer WHERE CustZip = \$X AND CustPhone LIKE \$Y%</i>	500/día
13. <i>SELECT * FROM Trade WHERE TrdStatus = \$X AND TrdDate = \$Y</i>	10/día
14. <i>SELECT * FROM Asset WHERE AssetName LIKE \$X%</i>	500/día

---

13. Para la carga de trabajo del problema 12, ¿existe alguna sentencia SELECT en la que el DBA quiera utilizar el optimizador de pistas? Por favor explique el tipo de pista que podría emplearse y su razonamiento para ello.
14. Investigue las herramientas para manejar los planes de acceso de un BDMS empresarial. Debe investigar las herramientas para el despliegue textual de los planes de acceso, el despliegue gráfico de los mismos y las pistas que ejerzan influencia en el juicio del optimizador.
15. Investigue las herramientas de diseño de bases de datos de un DBMS empresarial o herramienta CASE. Debe investigar las herramientas para el nivel de comando para la selección de índices, perfiles de tablas y perfiles de aplicación.
16. Investigue el componente de optimización de la consulta para un DBMS empresarial o herramienta CASE. Debe investigar los métodos de acceso para el acceso sencillo de tablas, los algoritmos de *join* y el uso de las estadísticas del optimizador.
17. Muestre el estado del Btree que aparece en la figura 8P.1 después de insertar las siguientes llaves: 115, 142, 111, 134, 170, 175, 127, 137, 108 y 140. El Btree tiene una capacidad máxima de llaves de 4. Muestre la división de nodos que ocurre cuando se insertan las llaves. Puede recibir ayuda para este problema por medio del uso de la herramienta interactiva de Btree que aparece en el sitio web <http://sky.fit.qut.edu.au/~maire/baobab/baobab.html>
18. De acuerdo con el problema 17, muestre el estado del Btree después de borrar las siguientes llaves: 108, 111 y 137. Muestre los encadenamientos del nodo y los préstamos de claves después de borrar las llaves. Puede recibir ayuda para este problema por medio del uso de la herramienta interactiva de Btree que aparece en el sitio web <http://sky.fit.qut.edu.au/~maire/baobab/baobab.html>

**FIGURA 8P.1**  
Btree inicial antes de las inserciones y eliminaciones



## Referencias para ampliar su estudio

El tema del diseño físico de bases de datos puede ser mucho más detallado y matemático de lo que se describe en este capítulo. Para una descripción más detallada de las estructuras de archivos y diseño físico de la base de datos, consulte libros de ciencias de la computación como Elmasri y Navathe (2004), y Teorey (1999). Para obtener un tutorial detallado acerca de la optimización de consultas, revise a Chaudhuri (1998), Jarke y Koch (1984), y Mannino, Chu y Sager (1988). Finkelstein, Schkolnick y Tiberio (1988) describen DBDSGN, una herramienta de selección de índices para SQL/DS, un DBMS relacional de IBM. Chaudhuri y Narasayya (1997, 2001) describen herramientas para la selección de índices y administración de estadísticas de Microsoft SQL Server. Shasha y Bonnet (2003) ofrecen más detalles acerca de las decisiones que corresponden al diseño físico de la base de datos. Para obtener mayor información acerca del diseño físico de la base de datos para un DBMS en particular, es necesario que consultar la documentación en línea para cada uno de los productos en específico. La sección diseño físico de bases de datos de la lista en línea de los recursos web, proporciona vínculos a herramientas de diseño físico de la base de datos y fuentes de consejos prácticos acerca del diseño físico de bases de datos.



# Desarrollo de aplicaciones con bases de datos relacionales

---

Parte

5

La parte 5 aporta los elementos para crear aplicaciones de bases de datos mediante conceptos y habilidades para la formulación avanzada de consultas (query), la especificación de requerimientos de datos para formularios y reportes de datos, así como los procedimientos almacenados y disparadores. El capítulo 9 amplía la formulación de la tabla avanzada que utiliza partes adicionales de SQL SELECT. El capítulo 10 describe el objetivo, la definición y el uso de vistas relacionales además de la especificación de requerimientos para la formulación y el reporte de datos. El capítulo 11 presenta los conceptos de la programación de base de datos y las prácticas de codificación para los procedimientos almacenados y disparadores en Oracle PL/SQL para apoyar el de los usos de la base de datos

---

**Capítulo 9.** Formulación avanzada de consultas con SQL

**Capítulo 10.** Desarrollo de aplicaciones con vistas

**Capítulo 11.** Procedimientos almacenados y disparadores



# Capítulo

# 9

# Formulación avanzada de consultas con SQL

## Objetivos de aprendizaje

Este capítulo amplía sus conocimientos en la formulación de consultas (query), pues explica problemas avanzados de concordancia de tablas que implican operadores de enlace externo, diferencia y división. También se demuestran otras partes de la sentencia SELECT (seleccionar) para explicar problemas avanzados de concordancia. Además, se explican los efectos sutiles de los valores nulos para ayudarle a interpretar los resultados de la consulta que implican valores nulos. Al finalizar este capítulo habrá adquirido los siguientes conocimientos y habilidades:

- Reconocer las consultas empaquetadas del tipo I para enlaces y entender el proceso asociado de evaluación conceptual.
- Reconocer las consultas empaquetadas del tipo II para enlaces y comprender el proceso asociado de evaluación conceptual.
- Reconocer los problemas que implican los operadores de enlace externo, diferencia y división.
- Adaptar ejemplos de enunciados en SQL para combinarlos con problemas que implican los operadores de enlace externo, diferencia y división.
- Entender el efecto de los valores nulos sobre condiciones, cálculos de conjuntos y agrupación.

## Panorama general

Al igual que en el primer capítulo de la parte 5 de este libro, el que nos ocupa se basa en el material trabajado en el capítulo 4. En él usted aprendió los fundamentos para la formulación de consultas usando SQL; además aprendió un importante subconjunto de la sentencia SELECT y su uso en problemas que implican enlace y agrupación. Este capítulo amplía su conocimiento en la formulación de consultas para problemas de combinaciones avanzadas. Para resolver estos problemas introducimos partes adicionales del enunciado SELECT.

Este capítulo continúa con los planteamientos de aprendizaje del capítulo 4: ofrece muchos ejemplos para imitarlos, así como lineamientos de solución de problemas para ayudarle a razonar problemas difíciles. Primero, aprenderá a formular problemas que implican el operador de enlace externo (*outer join*) utilizando nuevas palabras clave en la cláusula FROM (de); enseguida aprenderá a reconocer consultas empaquetadas y las aplicará para formular problemas que implican los operadores de enlace y diferencia. Luego aprenderá a reconocer problemas que

comprenden el operador de división y a formularlos empleando la cláusula GROUP BY (agrupar por), peticiones empaquetadas en la cláusula HAVING (teniendo) y la función COUNT (contar). Por último, aprenderá el efecto de los valores nulos en las condiciones simples, o bien en las compuestas con operadores lógicos, cálculos conjuntos y agrupación.

Este capítulo aborda características adicionales de Core SQL:2003, en especial, aquellas que no forman parte de SQL-92. Todos los ejemplos se ejecutan en versiones recientes de Microsoft Access (2002 y posteriores) y Oracle (9i y posteriores), excepto donde se indica.

## 9.1 Problemas de enlace externo (outer join)

Uno de los aspectos más poderosos, pero en ocasiones confusos, del enunciado SELECT es el número de maneras de expresar un enlace. En el capítulo 4, formuló enlaces usando el estilo del producto cruz y el estilo de operador de enlace. En el estilo de producto cruz, enlistó las tablas con la cláusula FROM y las condiciones de enlace con la cláusula WHERE (donde), utilizando las palabras clave INNER JOIN (enlace interno) y ON (en).

La principal ventaja del estilo del operador de enlace es que se pueden formular problemas que incluyen el operador de enlace externo. No es posible formular problemas de enlace externo con el estilo de producto cruz, excepto con extensiones propietarias de SQL. Esta sección demuestra el estilo del operador de enlace para problemas de enlace externo y combinaciones de enlaces internos y externos. Por otro lado, en el apéndice 9.C se presenta la extensión de enlace externo propietario de las versiones más antiguas de Oracle (8i y versiones previas). Para su referencia, se repite el diagrama de entidad-relación de la base de datos universitaria del capítulo 4 (vea la figura 9.1).

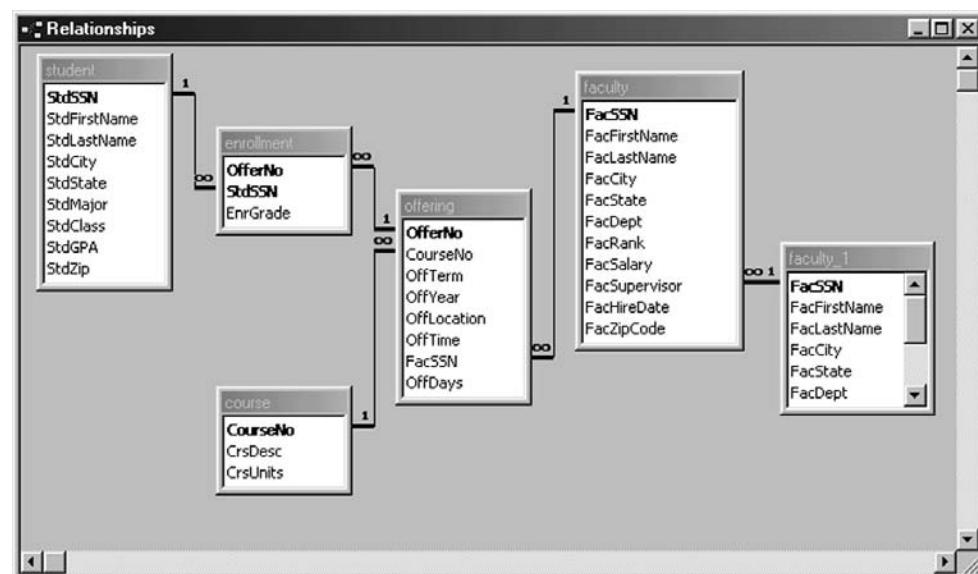
### enlace externo de un lado

un operador que genera el resultado de enlace (las filas de concordancia) más las filas de no concordancia de una de las tablas de entrada. SQL soporta el operador de enlace de un lado por medio de las palabras clave LEFT JOIN (enlace izquierdo) y RIGHT JOIN (enlace derecho).

### 9.1.1 Soporte de SQL para problemas de enlace externo

Un enlace entre dos tablas genera una tabla con filas que concuerdan en la(s) columna(s) de enlace. El operador de enlace externo (*outer join*) genera el resultado del enlace (las filas de concordancia) más las filas de no concordancia. Un enlace externo de un lado genera una nueva tabla con las filas de concordancia más las filas de no concordancia de *una* de las tablas. Por ejemplo, puede ser útil ver toda la lista de ofertas aun si una oferta no tiene un profesor asignado.

**FIGURA 9.1**  
Ventana de entidad-relación para la base de datos universitaria



SQL usa las palabras LEFT JOIN y RIGHT JOIN para producir un enlace de un lado.<sup>1</sup> La palabra clave LEFT JOIN crea una tabla de resultados que contiene las filas de concordancia y de no concordancia de la tabla izquierda. La palabra clave RIGHT JOIN crea una tabla de resultados que contiene las filas de concordancia y de no concordancia de la tabla derecha. Por consiguiente, el resultado de un enlace externo de un lado depende de la dirección (RIGHT o LEFT) y la posición de los nombres de la tabla. Los ejemplos 9.1 y 9.2 demuestran enlaces externos de un lado usando tanto la palabra clave LEFT como RIGHT. Las filas resultantes con valores en blanco para ciertas columnas son filas de no concordancia.

### EJEMPLO 9.1 (Access)

#### Enlace externo de un lado usando LEFT JOIN

Para las ofertas que comienzan con IS en el número de curso asociado, recupere el número de oferta, el número de curso, el número de profesor y el nombre del profesor. Incluya un curso en el resultado, incluso si todavía no se asigna al profesor. La contraparte de Oracle de este ejemplo utiliza % en lugar de \* como el carácter comodín.<sup>2</sup>

```
SELECT OfferNo, CourseNo, Offering.FacSSN, Faculty.FacSSN,
       FacFirstName, FacLastName
  FROM Offering LEFT JOIN Faculty
    ON Offering.FacSSN = Faculty.FacSSN
 WHERE CourseNo LIKE 'IS*'
```

OfferNo	CourseNo	Offering.FacSSN	Faculty.FacSSN	FacFirstName	FacLastName
1111	IS320				
2222	IS460				
1234	IS320	098-76-5432	098-76-5432	LEONARD	VINCE
3333	IS320	098-76-5432	098-76-5432	LEONARD	VINCE
4321	IS320	098-76-5432	098-76-5432	LEONARD	VINCE
4444	IS320	543-21-0987	543-21-0987	VICTORIA	EMMANUEL
8888	IS320	654-32-1098	654-32-1098	LEONARD	FIBON
9876	IS460	654-32-1098	654-32-1098	LEONARD	FIBON
5679	IS480	876-54-3210	876-54-3210	CRISTOPHER	COLAN
5678	IS480	987-65-4321	987-65-4321	JULIA	MILLS

### EJEMPLO 9.2 (Access)

#### Enlace externo de un lado usando RIGHT JOIN

Para las ofertas que comienzan con IS en el número de curso asociado, recupere el número de oferta, el número de curso, el número de profesor y el nombre del profesor. Incluya un curso en el resultado, incluso si todavía no se asigna al profesor. El resultado es idéntico al del ejemplo 9.1. La contraparte de este ejemplo en Oracle utiliza % en lugar de \* como el carácter comodín.

```
SELECT OfferNo, CourseNo, Offering.FacSSN, Faculty.FacSSN,
       FacFirstName, FacLastName
  FROM Faculty RIGHT JOIN Offering
    ON Offering.FacSSN = Faculty.FacSSN
 WHERE CourseNo LIKE 'IS*'
```

<sup>1</sup> Las palabras clave completas son LEFT OUTER JOIN y RIGHT OUTER JOIN. El estándar SQL:2003 y la mayoría de los DBMS permiten la omisión de la palabra clave OUTER.

<sup>2</sup> El apéndice 9.C muestra la notación propietaria de Oracle 8i para enlaces externos.

**enlace externo****completo**

un operador que genera el resultado del enlace (las filas de concordancia) más las filas de no concordancia de ambas tablas de entrada. SQL ofrece soporte para el operador de enlace externo completo (*full outer join*) a través de la palabra clave FULL JOIN.

Un enlace externo completo genera una tabla con las filas de concordancia más las filas de no concordancia de ambas tablas. Por lo regular, un enlace externo completo se usa para combinar dos tablas similares pero no compatibles en el enlace. Por ejemplo, las tablas *Student* y *Faculty* son similares porque contienen información acerca de las personas de la universidad. Sin embargo, no son compatibles en el enlace. Tienen columnas comunes como apellido, ciudad y número de seguridad social, pero también columnas únicas como GPA y salario. En ocasiones, necesitará escribir una consulta que combine ambas tablas. Por ejemplo, busque a todas las personas de la universidad en una ciudad determinada. En este tipo de problemas se utiliza un enlace externo completo.

SQL:2003 ofrece la palabra clave FULL JOIN, como se muestra en el ejemplo 9.3. Observe los valores nulos en ambas mitades del resultado (*Student* y *Faculty*).

**EJEMPLO 9.3****(SQL:2003 y Oracle 9i y posterior)****Enlace externo completo**

Combine las tablas *Faculty* y *Student* utilizando un enlace externo completo. Incluya el número de seguridad social, el nombre (nombre y apellido), el salario (sólo profesores) y GPA (sólo estudiantes) en el resultado. Este enunciado SQL no se ejecuta en Microsoft Access.

```
SELECT FacSSN, FacFirstName, FacLastName, FacSalary,
       StdSSN, StdFirstName, StdLastName, StdGPA
  FROM Faculty FULL JOIN Student
    ON Student.StdSSN = Faculty.FacSSN
```

FacSSN	FacFirstName	FacLastName	FacSalary	StdSSN	StdFirstName	StdLastName	StdGPA
				123456789	HOMER	WELLS	3
				124567890	BOB	NORBERT	2.7
				234567890	CANDY	KENDALL	3.5
				345678901	WALLY	KENDALL	2.8
				456789012	JOE	ESTRADA	3.2
				567890123	MARIAH	DODGE	3.6
				678901234	TESS	DODGE	3.3
				789012345	ROBERTO	MORALES	2.5
				890123456	LUKE	BRAZZI	2.2
				901234567	WILLIAM	PILGRIM	3.8
098765432	LEONARD	VINCE	35000				
543210987	VICTORIA	EMMANUEL	120000				
654321098	LEONARD	FIBON	70000				
765432109	NICKI	MACON	65000				
876543210	CRISTOPHER	COLAN	40000	876543210	CRISTOPHER	COLAN	4
987654321	JULIA	MILLS	75000				

Algunos DBMS (como Microsoft Access y Oracle 8i) no ofrecen soporte directo para el operador de enlace externo completo. En estos sistemas el enlace externo completo se formula tomando el enlace de dos enlaces externos de un lado siguiendo los pasos que se muestran a continuación. La sentencia SELECT que implementa estos pasos se muestra en el ejemplo 9.4. El apéndice 9.C contiene la contraparte de Oracle 8i para el ejemplo 9.4.

1. Construya un enlace derecho de *Faculty* y *Student* (filas de no concordancia de *Student*).
2. Construya un enlace izquierdo de *Faculty* y *Student* (filas de no concordancia de *Faculty*).
3. Construya un enlace de estas dos tablas temporales. Al usar el operador UNION recuerde que los dos argumentos de tabla deben ser “compatibles con los enlaces”: cada una de las columnas correspondientes de ambas tablas debe tener tipos de datos compatibles. De lo contrario, el operador UNION no va a funcionar como se esperaba.

**EJEMPLO 9.4  
(Access)****Enlace externo completo usando la unión de dos enlaces externos de un lado**

Combine las tablas *Faculty* y *Student* usando un enlace externo completo. Incluya el número de seguridad social, el nombre (nombre y apellido), el salario (sólo profesores) y GPA (sólo estudiantes) en el resultado. Éste es idéntico al del ejemplo 9.3.

```
SELECT FacSSN, FacFirstName, FacLastName, FacSalary,
       StdSSN, StdFirstName, StdLastName, StdGPA
  FROM Faculty RIGHT JOIN Student
    ON Student.StdSSN = Faculty.FacSSN
   UNION
SELECT FacSSN, FacFirstName, FacLastName, FacSalary,
       StdSSN, StdFirstName, StdLastName, StdGPA
  FROM Faculty LEFT JOIN Student
    ON Student.StdSSN = Faculty.FacSSN
```

**9.1.2 Combinación de enlaces internos y externos**

Los enlaces internos y externos se pueden combinar como se muestra en los ejemplos 9.5 y 9.6. Para facilitar la lectura, en general, es preferible utilizar el estilo de operador de enlaces, en lugar de combinarlo con el estilo de productos cruz.

**EJEMPLO 9.5  
(Access)****Combinación de un enlace externo de un lado y un enlace interno**

Combine las columnas de las tablas *Faculty*, *Offering* y *Course* para los cursos de sistemas de información (IS al principio del número de curso) que se ofrecen en 2006. Incluya una fila en el resultado, aun cuando no haya un profesor asignado. La contraparte de Oracle en este ejemplo utiliza % en lugar de \* como el carácter comodín.

```
SELECT OfferNo, Offering.CourseNo, OffTerm, CrsDesc,
       Faculty.FacSSN, FacFirstName, FacLastName
  FROM ( Faculty RIGHT JOIN Offering
        ON Offering.FacSSN = Faculty.FacSSN )
  INNER JOIN Course
    ON Course.CourseNo = Offering.CourseNo
 WHERE Course.CourseNo LIKE 'IS%' AND OffYear = 2006
```

OfferNo	CourseNo	OffTerm	CrsDesc	FacSSN	FacFirstName	FacLastName
1111	IS320	SUMMER	FUNDAMENTALS OF BUSINESS PROGRAMMING			
3333	IS320	SPRING	FUNDAMENTALS OF BUSINESS PROGRAMMING	098-76-5432	LEONARD	VINCE
4444	IS320	WINTER	FUNDAMENTALS OF BUSINESS PROGRAMMING	543-21-0987	VICTORIA	EMMANUEL
5678	IS480	WINTER	FUNDAMENTALS OF DATABASE MANAGEMENT	987-65-4321	JULIA	MILLS
5679	IS480	SPRING	FUNDAMENTALS OF DATABASE MANAGEMENT	876-54-3210	CRISTOPHER	COLAN
8888	IS320	SUMMER	FUNDAMENTALS OF BUSINESS PROGRAMMING	654-32-1098	LEONARD	FIBON
9876	IS460	SPRING	SYSTEMS ANALYSIS	654-32-1098	LEONARD	FIBON

**EJEMPLO 9.6  
(Access)****Combinación de un enlace externo de un lado con dos enlaces internos**

Incluya las filas de la tabla *Offering* en las que haya por lo menos un estudiante inscrito, además de los requisitos del ejemplo 9.5. Elimine las filas que se repiten cuando hay más de un estudiante inscrito en el curso ofrecido. La contraparte de este ejemplo en Oracle utiliza % en lugar de \* como el carácter comodín.

```
SELECT DISTINCT Offering.OfferNo, Offering.CourseNo,
   OffTerm, CrsDesc, Faculty.FacSSN, FacFirstName,
   FacLastName
  FROM ( ( Faculty RIGHT JOIN Offering
            ON Offering.FacSSN = Faculty.FacSSN )
    INNER JOIN Course
            ON Course.CourseNo = Offering.CourseNo )
    INNER JOIN Enrollment
            ON Offering.OfferNo = Enrollment.OfferNo
 WHERE Offering.CourseNo LIKE 'IS*' AND OffYear = 2006
```

OfferNo	CourseNo	OffTerm	CrsDesc	FacSSN	FacFirstName	FacLastName
5678	IS480	WINTER	FUNDAMENTALS OF DATABASE MANAGEMENT	987-65-4321	JULIA	MILLS
5679	IS480	SPRING	FUNDAMENTALS OF DATABASE MANAGEMENT	876-54-3210	CRISTOPHER	COLAN
9876	IS460	SPRING	SYSTEMS ANALYSIS	654-32-1098	LEONARD	FIBON

Al mezclar enlaces internos y externos deberá cuidar el orden en que combina las operaciones. Algunos DBMS, como Microsoft Access, señalan que los enlaces externos deben preceder a los internos. En los ejemplos 9.5 y 9.6, las operaciones de enlace externo de un lado preceden a las operaciones de enlace interno, como lo indica la posición del paréntesis. Sin embargo, las indicaciones de la documentación de Access no siempre se ponen en práctica. Así, el ejemplo 9.6a produce los mismos resultados que el ejemplo 9.6.

**EJEMPLO 9.6a  
(Access)****Combinación de un enlace externo de un lado y dos enlaces internos con el último enlace externo**

Incluya las filas de la tabla *Offering* donde hay por lo menos un estudiante inscrito, además de los requisitos del ejemplo 9.5. Elimine las filas que se repiten en las que hay más de un estudiante inscrito en un curso. La contraparte de este ejemplo en Oracle usa % en lugar de \* como el carácter comodín. El resultado es idéntico al del ejemplo 9.6.

```
SELECT DISTINCT Offering.OfferNo, Offering.CourseNo,
   OffTerm, CrsDesc, Faculty.FacSSN, FacFirstName,
   FacLastName
  FROM Faculty RIGHT JOIN
    ( ( Offering INNER JOIN Course
            ON Course.CourseNo = Offering.CourseNo )
    INNER JOIN Enrollment
            ON Offering.OfferNo = Enrollment.OfferNo )
            ON Offering.FacSSN = Faculty.FacSSN
 WHERE Offering.CourseNo LIKE 'IS*' AND OffYear = 2006
```

## 9.2 Entendiendo las consultas empaquetadas

**consulta empaquetada tipo I**  
una consulta empaquetada en la que la consulta interna no hace referencia a ninguna de las tablas utilizadas en la consulta externa. Las consultas empaquetadas tipo I se pueden usar para algunos problemas de enlace y de diferencia.

Una consulta empaquetada o subconsulta es una consulta (sentencia SELECT) dentro de otra. Por lo regular, este tipo de consulta aparece como parte de una condición en las cláusulas WHERE o HAVING. Las consultas empaquetadas también se pueden usar en la cláusula FROM. Las consultas empaquetadas llegan a utilizarse como un procedimiento (consulta empaquetada tipo I) en el que dicha consulta se ejecuta una vez, o como un ciclo (consulta empaquetada tipo II) en el que la consulta se ejecuta varias veces. Esta sección muestra ejemplos de ambos tipos de consulta empaquetada y explica los problemas en los que se pueden aplicar.

### 9.2.1 Consultas empaquetadas tipo I

Las consultas empaquetadas tipo I son como los procedimientos en un lenguaje de programación. Una consulta empaquetada tipo I se evalúa *una vez* y produce una tabla. La consulta empaquetada (o interna) no hace referencia a la consulta externa. Al utilizar el operador de comparación IN, una consulta empaquetada tipo I se puede usar para expresar un enlace. En el ejemplo 9.7, la consulta empaquetada en la tabla *Enrollment* genera una lista de valores de números de seguridad social que califican. Se selecciona una fila en la consulta externa sobre *Student* si el número de seguridad social es un elemento del resultado de la consulta empaquetada.

#### EJEMPLO 9.7

#### Empleo de una consulta empaquetada tipo I para realizar un enlace

Incluya el número de seguridad social, el nombre y la especialidad de los estudiantes que tienen una calificación alta ( $\geq 3.5$ ) en un curso ofrecido.

```
SELECT StdSSN, StdFirstName, StdLastName, StdMajor
  FROM Student
 WHERE Student.StdSSN IN
    ( SELECT StdSSN FROM Enrollment
      WHERE EnrGrade >= 3.5 )
```

StdSSN	StdFirstName	StdLastName	StdMajor
123-45-6789	HOMER	WELLS	IS
124-56-7890	BOB	NORBERT	FIN
234-56-7890	CANDY	KENDALL	ACCT
567-89-0123	MARIAH	DODGE	IS
789-01-2345	ROBERTO	MORALES	FIN
890-12-3456	LUKE	BRAZZI	IS
901-23-4567	WILLIAM	PILGRIM	IS

Las consultas empaquetadas tipo I sólo se deben usar cuando el resultado no contenga ninguna columna de las tablas en la consulta empaquetada. En el ejemplo 9.7, no se usa ninguna columna de la tabla *Enrollment* en el resultado. En el ejemplo 9.8, el enlace entre las tablas *Student* y

#### EJEMPLO 9.8

#### Combinación de una consulta empaquetada tipo I y el estilo operador de unión

Obtenga el nombre, la ciudad y la calificación de los estudiantes que tienen una calificación alta ( $\geq 3.5$ ) en un curso ofrecido a lo largo del año 2005.

```
SELECT StdFirstName, StdLastName, StdCity, EnrGrade
  FROM Student INNER JOIN Enrollment
    ON Student.StdSSN = Enrollment.StdSSN
 WHERE EnrGrade >= 3.5 AND Enrollment.OfferNo IN
    ( SELECT OfferNo FROM Offering
      WHERE OffTerm = 'FALL' AND OffYear = 2005 )
```

StdFirstName	StdLastName	StdCity	EnrGrade
CANDY	KENDALL	TACOMA	3.5
MARIAH	DODGE	SEATTLE	3.8
HOMER	WELLS	SEATTLE	3.5
ROBERTO	MORALES	SEATTLE	3.5

*Enrollment* no se pueden realizar con una consulta empaquetada tipo I porque *EnrGrade* aparece en el resultado.

Es posible tener varios niveles de consultas empaquetadas, aunque esta práctica no es recomendable porque puede ser difícil leer los enunciados. En la consulta empaquetada quizás haya otra que utiliza el operador de comparación IN en la cláusula WHERE. En el ejemplo 9.9, la consulta empaquetada de la tabla *Offering* tiene una de la tabla *Faculty*. No se necesita ninguna columna *Faculty* en la consulta principal ni en la empaquetada de *Offering*.

### EJEMPLO 9.9

#### Empleo de una consulta empaquetada tipo I dentro de otra consulta empaquetada tipo I

Obtenga el nombre, la ciudad y la calificación de los estudiantes que tienen una calificación alta ( $\geq 3.5$ ) en un curso ofrecido a lo largo del año 2005, que haya sido impartido por Leonard Vince.

```
SELECT StdFirstName, StdLastName, StdCity, EnrGrade
  FROM Student, Enrollment
 WHERE Student.StdSSN = Enrollment.StdSSN
       AND EnrGrade >= 3.5 AND Enrollment.OfferNo IN
        ( SELECT OfferNo FROM Offering
          WHERE OffTerm = 'FALL' AND OffYear = 2005
                AND FacSSN IN
                 ( SELECT FacSSN FROM Faculty
                   WHERE FacFirstName = 'LEONARD'
                         AND FacLastName = 'VINCE' ) )
```

StdFirstName	StdLastName	StdCity	EnrGrade
CANDY	KENDALL	TACOMA	3.5
MARIAH	DODGE	SEATTLE	3.8
HOMER	WELLS	SEATTLE	3.5
ROBERTO	MORALES	SEATTLE	3.5

El estilo tipo I da una idea visual de una consulta. Puede visualizar una subconsulta tipo I como si navevara entre tablas. Visite la tabla en la subconsulta para recopilar valores unidos que pueden utilizarse para seleccionar filas de la tabla en la consulta externa. El uso de consultas empaquetadas tipo I es, en gran medida, una cuestión de preferencias. Aun cuando usted no utilice este estilo de enlace, debe estar preparado para interpretar las consultas que otros escriben con consultas empaquetadas tipo I.

La sentencia DELETE (eliminar) ofrece otro uso para una consulta empaquetada tipo I. Una consulta empaquetada tipo I es útil cuando las filas eliminadas se relacionan con otras, como se muestra en el ejemplo 9.10. El uso de una consulta empaquetada tipo I es la forma estándar de hacer referencia a tablas relacionadas en las sentencias DELETE. En el capítulo 4 se mostró el estilo del operador de enlace dentro de una sentencia DELETE, una extensión

**EJEMPLO 9.10****Sentencia DELETE usando una consulta empaquetada tipo I**

Elimine los cursos impartidos por Leonard Vince. Se van a borrar tres filas *Offering*. Además, esta sentencia elimina las filas relacionadas en la tabla *Enrollment*, porque la cláusula ON DELETE está programada en CASCADE (cascada).

```
DELETE FROM Offering
WHERE Offering.FacSSN IN
( SELECT FacSSN FROM Faculty
  WHERE FacFirstName = 'LEONARD'
    AND FacLastName = 'VINCE' )
```

**EJEMPLO 9.11  
(sólo Access)****Enunciado DELETE con el uso de la operación INNER JOIN**

Elimine los cursos impartidos por Leonard Vince. Se van a borrar tres filas *Offering*. Además, este enunciado elimina las filas relacionadas en la tabla *Enrollment*, porque la cláusula ON DELETE está programada en CASCADE.

```
DELETE Offering.*
  FROM Offering INNER JOIN Faculty
    ON Offering.FacSSN = Faculty.FacSSN
  WHERE Faculty.FacFirstName = 'LEONARD'
    AND Faculty.FacLastName = 'VINCE'
```

propietaria de Microsoft Access. A manera de referencia, el ejemplo 9.11 muestra una sentencia DELETE utilizando el estilo del operador de enlace (*join*) que elimina las mismas filas que el ejemplo 9.10.

**problemas de  
diferencia**

los enunciados de problema que comprenden el operador de diferencia a menudo tienen dos nombres *no* relacionados en una oración. Por ejemplo, *los estudiantes que no son profesores y los empleados que no son clientes* son enunciados de problema que involucran un operador de diferencia.

**9.2.2 Formulaciones SQL limitadas para problemas de diferencia**

Recordemos que en el capítulo 3 establecimos que el operador de diferencia combina tablas al encontrar las filas de una primera tabla que no están en una segunda. Un uso típico del operador de diferencia es combinar dos tablas con algunas columnas similares, pero no totalmente compatibles con el enlace. Por ejemplo, tal vez quiera encontrar profesores que no son estudiantes. Aunque las tablas *Faculty* y *Student* contienen algunas columnas compatibles, las tablas no son compatibles con el enlace. La ubicación de la palabra *no* en el enunciado del problema indica que el resultado sólo contiene filas en la tabla *Faculty* y no en la tabla *Student*. Este requisito comprende una operación de diferencia.

Algunos problemas de diferencia se pueden formular usando una consulta empaquetada tipo I con el operador NOT IN (no en). Para poder usar esta consulta la comparación entre las tablas debe comprender una sola columna. En el ejemplo 9.12 se puede utilizar una consulta empaquetada tipo I porque la comparación sólo comprende una columna de la tabla *Faculty* (*FacSSN*).

**EJEMPLO 9.12****Uso de una consulta empaquetada tipo I para un problema de diferencia**

Obtenga el número de seguridad social, nombre (nombre y apellido), departamento y salario de los profesores que *no* son estudiantes.

```
SELECT FacSSN, FacFirstName, FacLastName, FacDept,
      FacSalary
    FROM Faculty
   WHERE FacSSN NOT IN
( SELECT StdSSN FROM Student )
```

FacSSN	FacFirstName	FacLastName	FacDept	FacSalary
098-76-5432	LEONARD	VINCE	MS	\$35 000.00
543-21-0987	VICTORIA	EMMANUEL	MS	\$120 000.00
654-32-1098	LEONARD	FIBON	MS	\$70 000.00
765-43-2109	NICKI	MACON	FIN	\$65 000.00
987-65-4321	JULIA	MILLS	FIN	\$75 000.00

Otra solución para algunos problemas de diferencia comprende un operador de enlace externo de un lado para generar una tabla sólo con filas de no concordancia. El operador de comparación IS NULL puede eliminar las filas con concordancia, como se demuestra en el ejemplo 9.13; sin embargo, esta formulación no se puede usar cuando hay condiciones que probar en la tabla que se excluye (*Student* en el ejemplo 9.13). Si hay condiciones que probar en la tabla *Student* (como estudiante en clase), es preciso emplear otro tipo de formulación SQL.

### EJEMPLO 9.13

#### Enlace externo de un lado con filas de no concordancia

Recupere el número de seguridad social, nombre, departamento y salario de los profesores que *no* son estudiantes. El resultado es idéntico al ejemplo 9.12.

```
SELECT FacSSN, FacFirstName, FacLastName, FacSalary
      FROM Faculty LEFT JOIN Student
                    ON Faculty.FacSSN = Student.StdSSN
      WHERE Student.StdSSN IS NULL
```

Aunque SQL:2003 tiene un operador de diferencia [la palabra clave EXCEPT (excepto)], en ocasiones no es conveniente su uso, porque en el resultado sólo se pueden mostrar las columnas comunes. El ejemplo 9.14 no ofrece el mismo resultado que el ejemplo 9.12 porque las columnas únicas de la tabla *Faculty* (*FacDept* y *FacSalary*) no están en el resultado. Es necesario formular otra consulta que usa el primer resultado para recuperar las columnas únicas de *Faculty*.

### EJEMPLO 9.14 (Oracle)

#### Consulta de diferencia

Muestre los profesores que *no* son estudiantes (sólo profesores). Muestre sólo las columnas comunes en el resultado. Observe que Microsoft Access no ofrece soporte para la palabra clave EXCEPT. Oracle utiliza la palabra clave MINUS (menos) en lugar de EXCEPT. El resultado es idéntico al ejemplo 9.12, a excepción de *FacCity* y *FacState* que aparecen en lugar de *FacDept* y *FacSalary*.

```
SELECT FacSSN AS SSN, FacFirstName AS FirstName,
      FacLastName AS LastName, FacCity AS City,
      FacState AS State
      FROM Faculty
      MINUS
      SELECT StdSSN AS SSN, StdFirstName AS FirstName,
      StdLastName AS LastName, StdCity AS City,
      StdState AS State
      FROM Student
```

*Los problemas de diferencia no se pueden resolver con enlaces de desigualdad*

Es importante notar que los problemas de diferencia, como el ejemplo 9.12, no se pueden resolver con un solo enlace. El ejemplo 9.12 requiere una búsqueda en todas las filas de la tabla *Student* para seleccionar una fila de profesores. En contraste, un enlace selecciona una fila de profesores al encontrar la primera fila de estudiantes con concordancia. Para comparar los problemas de diferencia y de enlace, analice el ejemplo 9.15. Aunque parece correcto, no presenta el resultado deseado. Todas las filas de profesores van a estar en el resultado porque hay por lo menos una fila de estudiantes que no concuerda con todas las filas de profesores.

### EJEMPLO 9.15

#### Enlace de desigualdad

Formulación errónea del problema “Recupere el número de seguridad social, nombre (nombre y apellido) y rango de los profesores que *no* son estudiantes”. El resultado contiene todas las filas de profesores.

```
SELECT DISTINCT FacSSN, FacFirstName, FacLastName, FacRank
  FROM Faculty, Student
 WHERE Student.StdSSN <> Faculty.FacSSN
```

Para entender el ejemplo 9.15 puede emplear el proceso de evaluación conceptual que estudiamos en el capítulo 4 (sección 4.3). Las tablas de resultados muestran el producto cruz (tabla 9.3) de las tablas 9.1 y 9.2, seguidas por las filas que satisfacen la condición WHERE (tabla 9.4). Observe que sólo se elimina una fila del producto cruz. El resultado final (tabla 9.5) contiene todas las filas de la tabla 9.2.

**TABLA 9.1**  
Ejemplo de la tabla  
*Student*

StdSSN	StdFirstName	StdLastName	StdMajor
123-45-6789	HOMER	WELLS	IS
124-56-7890	BOB	NORBERT	FIN
234-56-7890	CANDY	KENDALL	ACCT

**TABLA 9.2**  
Ejemplo de la tabla  
*Faculty*

FacSSN	FacFirstName	FacLastName	FacRank
098-76-5432	LEONARD	VINCE	ASST
543-21-0987	VICTORIA	EMMANUEL	PROF
876-54-3210	CRISTOPHER	COLAN	ASST

**TABLA 9.3** Producto cruz de las tablas de ejemplo *Student* y *Faculty*

FacSSN	FacFirstName	FacLastName	FacRank	StdSSN	StdFirstName	StdLastName	StdMajor
098-76-5432	LEONARD	VINCE	ASST	123-45-6789	HOMER	WELLS	IS
098-76-5432	LEONARD	VINCE	ASST	124-56-7890	BOB	NORBERT	FIN
098-76-5432	LEONARD	VINCE	ASST	876-54-3210	CRISTOPHER	COLAN	IS
543-21-0987	VICTORIA	EMMANUEL	PROF	123-45-6789	HOMER	WELLS	IS
543-21-0987	VICTORIA	EMMANUEL	PROF	124-56-7890	BOB	NORBERT	FIN
543-21-0987	VICTORIA	EMMANUEL	PROF	876-54-3210	CRISTOPHER	COLAN	IS
876-54-3210	CRISTOPHER	COLAN	ASST	123-45-6789	HOMER	WELLS	IS
876-54-3210	CRISTOPHER	COLAN	ASST	124-56-7890	BOB	NORBERT	FIN
876-54-3210	CRISTOPHER	COLAN	ASST	876-54-3210	CRISTOPHER	COLAN	IS

**TABLA 9.4** Restricción de la tabla 9.3 para eliminar las filas con concordancia

FacSSN	FacFirstName	FacLastName	FacRank	StdSSN	StdFirstName	StdLastName	StdMajor
098-76-5432	LEONARD	VINCE	ASST	123-45-6789	HOMER	WELLS	IS
098-76-5432	LEONARD	VINCE	ASST	124-56-7890	BOB	NORBERT	FIN
098-76-5432	LEONARD	VINCE	ASST	876-54-3210	CRISTOPHER	COLAN	IS
543-21-0987	VICTORIA	EMMANUEL	PROF	123-45-6789	HOMER	WELLS	IS
543-21-0987	VICTORIA	EMMANUEL	PROF	124-56-7890	BOB	NORBERT	FIN
543-21-0987	VICTORIA	EMMANUEL	PROF	876-54-3210	CRISTOPHER	COLAN	IS
876-54-3210	CRISTOPHER	COLAN	ASST	123-45-6789	HOMER	WELLS	IS
876-54-3210	CRISTOPHER	COLAN	ASST	124-56-7890	BOB	NORBERT	FIN

**TABLA 9.5**  
Proyección de la  
tabla 9.4 para eli-  
minar las columnas  
*Student*

FacSSN	FacFirstName	FacLastName	FacRank
098-76-5432	LEONARD	VINCE	ASST
543-21-0987	VICTORIA	EMMANUEL	PROF
876-54-3210	CRISTOPHER	COLAN	ASST

**TABLA 9.6**  
Limitaciones de las  
formulaciones SQL  
para los problemas de  
diferencia

Formulación SQL	Limitaciones
Consulta empaquetada tipo I con el operador NOT IN	Sólo una columna para comparar las filas de las dos tablas
Enlace externo de un lado con una condición IS NULL (es nulo)	Sin condiciones (excepto la condición IS NULL) en la tabla que se excluye
Operación de diferencia con el uso de las palabras clave EXCEPT o MINUS	El resultado debe contener sólo columnas compatibles con el enlace

#### *Resumen de formulaciones limitadas para problemas de diferencia*

Esta sección ha estudiado tres formulaciones SQL para problemas de diferencia. Como indica la tabla 9.6, cada formulación tiene limitaciones. En la práctica, el enfoque de enlace externo de un lado es el más restrictivo, ya que muchos problemas comprenden condiciones sobre la tabla de exclusión. La sección 9.2.3 presenta una formulación más general sin las limitaciones que se observan en la tabla 9.6.

### 9.2.3 Uso de consultas empaquetadas tipo II para problemas de diferencia

Aunque las consultas empaquetadas tipo II ofrecen una solución más general para los problemas de diferencia, conceptualmente son más complejas que las consultas empaquetadas tipo I. Las consultas empaquetadas tipo II tienen dos características distintivas. En primer lugar, hacen referencia a una o más columnas desde una consulta externa. En ocasiones se conocen como subconsultas correlacionadas, porque hacen referencia a columnas que se utilizan en consultas externas. En contraste, las consultas empaquetadas tipo I no están correlacionadas con consultas externas. En el ejemplo 9.16, la consulta empaquetada contiene una referencia a la tabla *Faculty* usada en la consulta externa.

La segunda característica distintiva de las consultas empaquetadas tipo II comprende su ejecución. Una consulta de este tipo se ejecuta una vez para *cada* fila de la consulta externa. En este sentido, una consulta empaquetada tipo II es similar a un ciclo empaquetado que se ejecuta una vez por cada ejecución del ciclo externo. Cada ejecución del ciclo interno utiliza las variables del ciclo externo. En otras palabras, la consulta interna emplea uno o más valores de la consulta externa en cada ejecución.

Para ayudarle a comprender el ejemplo 9.16, la tabla 9.9 hace un seguimiento de la consulta empaquetada utilizando las tablas 9.7 y 9.8. El operador EXISTS es verdadero si la

#### **consulta empaquetada tipo II**

una consulta empaquetada en la que la consulta interna hace referencia a una tabla usada en la consulta externa. Como una consulta empaquetada tipo II se ejecuta para cada fila de la consulta externa, las consultas empaquetadas tipo II son más difíciles de entender y ejecutar que las de tipo I.

**TABLA 9.7**  
Ejemplo de la tabla  
*Faculty*

FacSSN	FacFirstName	FacLastName	FacRank
098-76-5432	LEONARD	VINCE	ASST
543-21-0987	VICTORIA	EMMANUEL	PROF
876-54-3210	CRISTOPHER	COLAN	ASST

**TABLA 9.8**  
Ejemplo de la tabla  
*Student*

StdSSN	StdFirstName	StdLastName	StdMajor
123-45-6789	HOMER	WELLS	IS
124-56-7890	BOB	NORBERT	FIN
876-54-3210	CRISTOPHER	COLAN	IS

**TABLA 9.9**  
Seguimiento de ejecución de la consulta empaquetada en el ejemplo 9.16

FacSSN	Result of subquery execution	NOT EXISTS
098-76-5432	0 rows retrieved	true
543-21-0987	0 rows retrieved	true
876-54-3210	1 row retrieved	false

### EJEMPLO 9.16

#### Uso de una consulta empaquetada tipo II para un problema de diferencia

Obtenga el número de seguridad social, nombre (nombre y apellido), departamento y salario de los profesores que *no* son estudiantes.

```
SELECT FacSSN, FacFirstName, FacLastName, FacDept, FacSalary
FROM Faculty
WHERE NOT EXISTS
( SELECT * FROM Student
WHERE Student.StdSSN = Faculty.FacSSN )
```

FacSSN	FacFirstName	FacLastName	FacDept	FacSalary
098-76-5432	LEONARD	VINCE	MS	\$35 000.00
543-21-0987	VICTORIA	EMMANUEL	MS	\$120 000.00
654-32-1098	LEONARD	FIBON	MS	\$70 000.00
765-43-2109	NICKI	MACON	FIN	\$65 000.00
987-65-4321	JULIA	MILLS	FIN	\$75 000.00

consulta empaquetada regresa una o más filas. En contraste, el operador NOT EXISTS es verdadero si la consulta empaquetada regresa 0 filas. Por lo tanto, sólo se selecciona una fila de profesores en la consulta externa si no hay filas de estudiantes en concordancia con la consulta empaquetada. Por ejemplo, se seleccionan las primeras dos filas en la tabla 9.7 porque no hay filas con concordancia en la tabla 9.8. La tercera fila *no* se selecciona porque la consulta empaquetada regresa una sola fila (la tercera de la tabla 9.7).

El ejemplo 9.17 muestra otra formulación que aclara el significado del operador NOT EXISTS (no existe). Aquí, se selecciona la fila de profesores si el número de filas de la consulta empaquetada es 0. Utilizando las tablas de muestra (9.7 y 9.8), el resultado de la consulta empaquetada es 0 para las primeras dos filas de profesores.

#### Problemas de diferencia más complicada

Los problemas de diferencia más complicada combinan una operación de diferencia con operaciones de enlace. Por ejemplo, considere la consulta para obtener a los estudiantes que tomaron todos los cursos de sistemas de información durante el invierno de 2006 con el mismo profesor. Los resultados de la consulta deben incluir tanto a los estudiantes que tomaron un solo curso, como a aquellos que tomaron más de uno.

#### operador NOT EXISTS

un operador de comparación de tablas que a menudo se usa con consultas empaquetadas tipo II. NOT EXISTS es verdadero para una fila en una consulta externa si la consulta interna no regresa ninguna fila, y falso si la consulta interna regresa una o más filas.

**EJEMPLO 9.17 Uso de una consulta empaquetada tipo II con la función COUNT**

Obtenga el número de seguridad social, nombre, departamento y salario de los profesores que no son estudiantes. El resultado es el mismo que en el ejemplo 9.16.

```
SELECT FacSSN, FacFirstName, FacLastName, FacDept,
       FacSalary
  FROM Faculty
 WHERE 0 =
    ( SELECT COUNT(*) FROM Student
      WHERE Student.StdSSN = Faculty.FacSSN )
```

- Elabore una lista de estudiantes que tomaron cursos de sistemas de información durante el invierno de 2006 (operación de enlace).
- Elabore otra lista de estudiantes que han tomado cursos de IS durante el invierno de 2006 con más de un profesor (operación enlace).
- Utilice una operación de diferencia (primera lista de estudiantes menos segunda lista de estudiantes) para producir el resultado.

El hecho de conceptuar un problema de esta manera le obliga a reconocer y comprender una operación de diferencia. Si reconoce la operación de diferencia, puede hacer una formulación

**EJEMPLO 9.18 Problema de diferencia más complicada con el uso  
(Access) de una consulta empaquetada tipo II**

Haga una lista con los números de seguridad social y los nombres de los estudiantes que tomaron todos los cursos de sistemas de información durante el invierno de 2006 con el mismo profesor. Incluya tanto a los que tomaron sólo un curso como a los que tomaron más de uno. Observe que en la consulta empaquetada, las columnas *Enrollment*, *StdSSN* y *Offering.FacSSN* se refieren a la consulta externa.

```
SELECT DISTINCT Enrollment.StdSSN, StdFirstName, StdLastName
  FROM Student, Enrollment, Offering
 WHERE Student.StdSSN = Enrollment.StdSSN
       AND Enrollment.OfferNo = Offering.OfferNo
       AND CourseNo LIKE 'IS*' AND OffTerm = 'WINTER'
       AND OffYear = 2006 AND NOT EXISTS
    ( SELECT * FROM Enrollment E1, Offering O1
      WHERE E1.OfferNo = O1.OfferNo
            AND Enrollment.StdSSN = E1.StdSSN
            AND O1.CourseNo LIKE 'IS*'
            AND O1.OffYear = 2006
            AND O1.OffTerm = 'WINTER'
            AND Offering.FacSSN <> O1.FacSSN )
```

StdSSN	StdFirstName	StdLastName
123-45-6789	HOMER	WELLS
234-56-7890	CANDY	KENDALL
345-67-8901	WALLY	KENDALL
456-78-9012	JOE	ESTRADA
567-89-123	MARIAH	DODGE

**EJEMPLO 9.18  
(Oracle)****Problema de diferencia más complicada con el uso de una consulta empaquetada tipo II**

Haga una lista con el número de seguridad social y el nombre de los estudiantes que tomaron todos los cursos de sistemas de información durante el invierno de 2006 con el mismo profesor. Incluya tanto a los que tomaron sólo un curso como a los que tomaron más de uno.

```
SELECT DISTINCT Enrollment.StdSSN, StdFirstName,
               StdLastName
      FROM Student, Enrollment, Offering
     WHERE Student.StdSSN = Enrollment.StdSSN
           AND Enrollment.OfferNo = Offering.OfferNo
           AND CourseNo LIKE 'IS%' AND OffTerm = 'WINTER'
           AND OffYear = 2006 AND NOT EXISTS
             ( SELECT * FROM Enrollment E1, Offering O1
               WHERE E1.OfferNo = O1.OfferNo
                 AND Enrollment.StdSSN = E1.StdSSN
                 AND O1.CourseNo LIKE 'IS%'
                 AND O1.OffYear = 2006
                 AND O1.OffTerm = 'WINTER'
                 AND Offering.FacSSN <> O1.FacSSN )
```

en SQL que comprenda una consulta empaquetada (tipo II con NOT EXISTS o tipo I con NOT IN) o la palabra clave EXCEPT. El ejemplo 9.18 muestra una solución NOT EXISTS en la que la consulta externa recupera una fila de estudiantes si el estudiante no tiene un curso con un profesor *diferente* que el de la consulta interna.

El ejemplo 9.19 utiliza el operador NOT EXISTS para solucionar un problema de diferencia compleja. En forma conceptual, este problema comprende una operación de diferencia entre dos conjuntos: el conjunto de todos los profesores y el conjunto de todos los profesores que

**EJEMPLO 9.19****Otro problema de diferencia con el uso de una consulta empaquetada tipo II**

Haga una lista con el nombre (nombre y apellido) y departamento de los profesores que *no* impartieron clases durante el invierno de 2006.

```
SELECT DISTINCT FacFirstName, FacLastName, FacDept
      FROM Faculty
     WHERE NOT EXISTS
       ( SELECT * FROM Offering
         WHERE Offering.FacSSN = Faculty.FacSSN
           AND OffTerm = 'WINTER' AND OffYear = 2006 )
```

FacFirstName	FacLastName	FacDept
CRISTOPHER	COLAN	MS
LEONARD	FIBON	MS
LEONARD	VINCE	MS

impartieron clases en el semestre que se especifica. La operación de diferencia se puede implementar seleccionando un profesor de la lista de la consulta externa si éste no impartió un curso durante el plazo que se especifica en el resultado de la consulta interna.

El ejemplo 9.20 muestra una vez más el uso del operador NOT EXISTS para solucionar un problema de diferencia compleja. En este problema, la palabra *only* (sólo) conecta distintas partes del enunciado e indica una operación de diferencia. En forma conceptual, este problema comprende una operación de diferencia entre dos conjuntos: un conjunto de profesores que impartieron clases durante el invierno de 2006, y un conjunto de profesores que impartieron clases en el invierno de 2006, además de hacerlo en otros semestres. La operación de diferencia se puede implementar seleccionando un profesor que haya impartido clases en invierno de 2006 en la consulta externa, si ese mismo profesor no impartió un curso en un semestre diferente en la consulta empaquetada.

### EJEMPLO 9.20

### Otro problema de diferencia con el uso de una consulta empaquetada tipo II

Haga una lista con el nombre (nombre y apellido) y departamento de los profesores que no sólo impartieron clases en el invierno de 2006.

```
SELECT DISTINCT FacFirstName, FacLastName, FacDept
  FROM Faculty F1, Offering O1
 WHERE F1.FacSSN = O1.FacSSN
   AND OffTerm = 'WINTER' AND OffYear = 2006
   AND NOT EXISTS
 ( SELECT * FROM Offering O2
   WHERE O2.FacSSN = F1.FacSSN
     AND ( OffTerm <> 'WINTER' OR OffYear <> 2006 ) )
```

FacFirstName	FacLastName	FacDept
EMMANUEL	VICTORIA	MS
MILLS	JULIA	FIN

#### 9.2.4 Consultas empaquetadas con la cláusula FROM

Hasta el momento ha empaquetado consultas con la cláusula WHERE y ciertos operadores de comparación (IN y EXISTS), así como con operadores de comparación tradicionales cuando la consulta empaquetada produce un solo valor, como la cantidad del número de filas. En la siguiente sección se demuestra que las consultas empaquetadas pueden aparecer con la cláusula HAVING de manera similar que cuando se usa la cláusula WHERE. Las consultas empaquetadas con las cláusulas WHERE y HAVING han sido parte del diseño de SQL desde un principio.

En contraste, las consultas empaquetadas con la cláusula FROM fueron una nueva extensión en SQL:1999. El diseño de SQL:1999 inició una filosofía de consistencia en el diseño del lenguaje. Consistencia significa que dondequiera que se permita un objeto también se permite la expresión del objeto. Esta filosofía aplicada a la cláusula FROM significa que siempre que una tabla está permitida, también lo está una expresión (una consulta empaquetada). Las consultas empaquetadas con la cláusula FROM no se utilizan con tanta frecuencia como las consultas empaquetadas con las cláusulas WHERE y HAVING. El resto de esta sección demuestra algunos usos especializados de las consultas empaquetadas para la cláusula FROM.

Uno de los usos de las consultas empaquetadas en la cláusula FROM consiste en calcular una función conjunta con otra función conjunta (conjuntos empaquetados). SQL no permite una función conjunta dentro de otra. Una consulta empaquetada en la cláusula FROM supera la prohibición en contra de los conjuntos empaquetados, como lo demuestra el ejemplo 9.21. Sin una consulta empaquetada en la cláusula FROM, serán necesarias dos consultas para producir el resultado. En Access, la consulta empaquetada sería una consulta almacenada. En Oracle, la consulta empaquetada sería una vista (vea el capítulo 10 para una explicación de las vistas).

**EJEMPLO 9.21****Uso de una consulta empaquetada en la cláusula FROM**

Haga una lista con el número de curso y su descripción, la cantidad de cursos y el promedio de inscripciones en todos los cursos.

```
SELECT T.CourseNo, T.CrsDesc, COUNT(*) AS NumOfferings,
       Avg(T.EnrollCount) AS AvgEnroll
  FROM
    ( SELECT Course.CourseNo, CrsDesc,
             Offering.OfferNo, COUNT(*) AS EnrollCount
      FROM Offering, Enrollment, Course
     WHERE Offering.OfferNo = Enrollment.OfferNo
       AND Course.CourseNo = Offering.CourseNo
    GROUP BY Course.CourseNo, CrsDesc, Offering.OfferNo
    ) T
 GROUP BY T.CourseNo, T.CrsDesc
```

CourseNo	CrsDesc	NumOfferings	AvgEnroll
FIN300	FUNDAMENTALS OF FINANCE	1	2
FIN450	PRINCIPLES OF INVESTMENTS	1	2
FIN480	CORPORATE FINANCE	1	3
IS320	FUNDAMENTALS OF BUSINESS PROGRAMMING	2	6
IS460	SYSTEMS ANALYSIS	1	7
IS480	FUNDAMENTALS OF DATABASE MANAGEMENT	2	5.5

Otro uso de una consulta empaquetada en la cláusula FROM es calcular conjuntos a partir de varias agrupaciones. Sin una consulta empaquetada en la cláusula FROM, una consulta puede contener conjuntos de una agrupación. Por ejemplo, se necesitan varias agrupaciones para resumir el número de estudiantes por curso y el número de recursos por curso. Esta consulta puede ser útil si el diseño de la base de datos de la universidad se extendió con una tabla *Resource* y una tabla asociada (*ResourceUsage*), conectadas a las tablas *Offering* y *Resource* a través de relaciones 1-M. La consulta necesitará de dos consultas empaquetadas en la cláusula FROM: una para recuperar la cuenta de inscripciones a los cursos y otra para recuperar la cantidad de recursos para cada curso.

En Access, una consulta empaquetada en la cláusula FROM puede compensar la incapacidad de usar la palabra clave DISTINCT (distinto) dentro de las funciones conjuntas. Por ejemplo, necesitamos la palabra clave DISTINCT para calcular el número de cursos diferentes

**EJEMPLO 9.22  
(Oracle)****Uso de la palabra clave DISTINCT dentro de la función COUNT**

Haga una lista con el número de seguridad social, el apellido y el número de cursos únicos impartidos.

```
SELECT Faculty.FacSSN, FacLastName,
       COUNT(DISTINCT CourseNo) AS NumPreparations
  FROM Faculty, Offering
 WHERE Faculty.FacSSN = Offering.FacSSN
 GROUP BY Faculty.FacSSN, FacLastName
```

FacSSN	FacLastName	NumPreparations
098-76-5432	VINCE	1
543-21-0987	EMMANUEL	1
654-32-1098	FIBON	2
765-43-2109	MACON	2
876-54-3210	COLAN	1
987-65-4321	MILLS	2

que imparten los profesores, como se muestra en el ejemplo 9.22. Para producir los mismos resultados en Access, es necesaria una consulta empaquetada en la cláusula FROM, como se muestra en el ejemplo 9.23. La consulta empaquetada en la cláusula FROM utiliza la palabra clave DISTINCT para eliminar números de curso repetidos. La sección 9.3.3 contiene ejemplos adicionales en los que se usan consultas empaquetadas en la cláusula FROM para compensar la palabra clave DISTINCT dentro de la función COUNT.

### EJEMPLO 9.23

#### Uso de una consulta empaquetada en la cláusula FROM en lugar de la palabra clave DISTINCT dentro de la función COUNT

Haga una lista con el número de seguridad social, el apellido y el número de cursos únicos impartidos. El resultado es idéntico al ejemplo 9.22. Aunque el enunciado SELECT se ejecuta en Access y Oracle, deberá usarlo en el ejemplo 9.22 con Oracle, ya que se ejecuta con más rapidez.

```
SELECT T.FacSSN, T.FacLastName,
       COUNT(*) AS NumPreparations
  FROM
    ( SELECT DISTINCT Faculty.FacSSN, FacLastName, CourseNo
      FROM Offering, Faculty
     WHERE Offering.FacSSN = Faculty.FacSSN ) T
 GROUP BY T.FacSSN, T.FacLastName
```

## 9.3 Formulación de problemas de división

Los problemas de división se encuentran entre los más difíciles. Debido a esta dificultad, revisamos en forma breve el operador dividir que vimos en el capítulo 3. Después de la revisión, esta sección estudia algunos problemas de división sencillos antes de pasar a otros más avanzados.

### 9.3.1 Revisión del operador dividir

Para revisar el operador dividir (divide) considere una base de datos universitaria simplificada que consta de tres tablas: *Student1* (tabla 9.10), *Club* (tabla 9.11) y *StdClub* (tabla 9.12), que muestra la membresía de los estudiantes a los clubes. Por lo regular, el operador dividir se aplica para unir tablas que muestran relaciones M-N. La tabla *StdClub* une las tablas *Student1* y *Club*: un estudiante puede pertenecer a varios clubes y un club puede tener varios estudiantes.

**TABLA 9.10**  
Listado de la tabla  
*Student1*

StdNo	SName	SCity
S1	JOE	SEATTLE
S2	SALLY	SEATTLE
S3	SUE	PORTLAND

**TABLA 9.11**  
Listado de la tabla  
*Club*

ClubNo	CName	CPurpose	CBudget	CActual
C1	DELTA	SOCIAL	\$1 000.00	\$1200.00
C2	BITS	ACADEMIC	\$500.00	\$350.00
C3	HELPS	SERVICE	\$300.00	\$330.00
C4	SIGMA	SOCIAL		\$150.00

**TABLA 9.12**  
Listado de la tabla  
*StdClub*

StdNo	ClubNo
S1	C1
S1	C2
S1	C3
S1	C4
S2	C1
S2	C4
S3	C3

### dividir

un operador de álgebra relacional que combina las filas de dos tablas. El operador dividir produce una tabla en la que los valores de una columna de una tabla de entrada se relacionan con todos los valores de una columna de la segunda tabla.

El operador dividir crea una tabla que consiste en los valores de una columna (*StdNo*) que compara *todos* los valores en la columna especificada (*ClubNo*) de una segunda tabla (*Club*). Un problema de división común es hacer una lista de los estudiantes que pertenecen a *todos* los clubes. La tabla resultante contiene sólo el estudiante S1, porque S1 está relacionado con los cuatro clubes.

La división es más difícil en el aspecto conceptual que el enlace, porque la primera compara todos los valores, mientras que la segunda sólo compara uno. Si este problema comprendiera un enlace, se establecería como “haga una lista de los estudiantes que pertenecen a *cualquier* club”. La diferencia clave es la palabra *cualquier* contra *todos*. La mayoría de los problemas de división se pueden escribir con los adjetivos *cada* o *todos* entre una frase con verbo que representa una tabla o un nombre que representa otra tabla. En este ejemplo, la frase “estudiantes que pertenecen a todos los clubes” se ajusta al patrón. Otro ejemplo es “estudiantes que han tomado todos los cursos”.

### 9.3.2 Problemas sencillos de división

Hay muchas formas de realizar una división en SQL. Algunos libros describen un enfoque que utiliza consultas empaquetadas tipo II. Como este enfoque puede ser difícil de entender si no ha tomado un curso de lógica, aquí empleamos un enfoque diferente que utiliza la función COUNT con una consulta empaquetada en la cláusula HAVING.

La idea básica es comparar el número de estudiantes relacionados con un club en la tabla *StdClub*, con el número de clubes en la tabla *Club*. Para realizar esta operación, agrupe la tabla *StdClub* en *StdNo* y compare el número de filas en cada grupo *StdNo* con el número de filas en la tabla *Club*. Puede hacer esta comparación utilizando una consulta empaquetada con la cláusula HAVING, como se muestra en el ejemplo 9.24.

#### EJEMPLO 9.24

#### El problema de división más sencillo

Haga una lista con el número de estudiantes que pertenecen a todos los clubes.

```
SELECT StdNo
  FROM StdClub
 GROUP BY StdNo
 HAVING COUNT(*) = ( SELECT COUNT(*) FROM Club )
```

StdNo
S1

Debemos hacer notar que el COUNT(\*) del lado izquierdo indica el número de filas en el grupo *StdNo*. El lado derecho contiene una consulta empaquetada con sólo un COUNT(\*) en el resultado. La consulta empaquetada es tipo I porque no hay conexión con la consulta externa; por tanto, la consulta empaquetada sólo se ejecuta una vez y regresa una sola fila con un valor (el número de filas en la tabla *Club*).

Ahora vamos a examinar algunas variaciones del primer problema. La más típica es recuperar a los estudiantes que pertenecen a un subconjunto de clubes, en lugar de todos los clubes. Por ejemplo, recupere a los estudiantes que pertenecen a todos los clubes sociales. Para lograr este cambio deberá modificar el ejemplo 9.24, al incluir una condición WHERE en la consulta interna y empaquetada. En lugar de contar todas las filas *Student1* en un grupo *StdNo*, cuente sólo las filas en las que el propósito del club es social. Compare esta cuenta con el número de clubes sociales en la tabla *Club*. El ejemplo 9.25 muestra estas modificaciones.

### EJEMPLO 9.25

#### Problema de división para encontrar una concordancia de subconjuntos

Haga una lista con el número de estudiante de aquellos que pertenecen a todos los clubes sociales.

```
SELECT StdNo
  FROM StdClub, Club
 WHERE StdClub.ClubNo = Club.ClubNo
   AND CPurpose = 'SOCIAL'
 GROUP BY StdNo
 HAVING COUNT(*) =
 ( SELECT COUNT(*) FROM Club
   WHERE CPurpose = 'SOCIAL' )
```

StdNo
S1
S2

En los ejemplos 9.26 y 9.27 se muestran otras variaciones. En el ejemplo 9.26 es necesario un enlace entre *StdClub* y *Student* para obtener el nombre del estudiante. El ejemplo 9.27 revierte los problemas anteriores al buscar clubes en lugar de estudiantes.

### EJEMPLO 9.26

#### Problema de división con enlaces

Haga una lista con el número de estudiante y el nombre de los alumnos que pertenezcan a todos los clubes sociales.

```
SELECT Student1.StdNo, SName
  FROM StdClub, Club, Student1
 WHERE StdClub.ClubNo = Club.ClubNo
   AND Student1.StdNo = StdClub.StdNo
   AND CPurpose = 'SOCIAL'
 GROUP BY Student1.StdNo, SName
 HAVING COUNT(*) =
 ( SELECT COUNT(*) FROM Club
   WHERE CPurpose = 'SOCIAL' )
```

StdNo	SName
S1	JOE
S2	SALLY

**EJEMPLO 9.27****Otro problema de división**

Haga una lista con los números de club que tienen como miembros a todos los estudiantes de Seattle.

```
SELECT ClubNo
  FROM StdClub, Student1
 WHERE Student1.StdNo = StdClub.StdNo
   AND SCity = 'SEATTLE'
GROUP BY ClubNo
HAVING COUNT(*) =
( SELECT COUNT(*) FROM Student1
  WHERE SCity = 'SEATTLE' )
```

ClubNo
C1
C4

**9.3.3 Problemas de división avanzados**

El ejemplo 9.28 (utilizando las tablas de bases de datos universitarias originales) ilustra otra complicación en los problemas de división con SQL. Antes de estudiar esta complicación adicional, analicemos un problema más sencillo. El ejemplo 9.28 se puede formular con la misma técnica que se muestra en la sección 9.3.2. Primero, una las tablas *Faculty* y *Offering*, seleccione las filas con concordancia en las condiciones WHERE y agrupe el resultado por nombre de profesor (nombre y apellido). Luego, compare la cuenta de las filas en cada grupo *nombre* de profesor con el número de cursos de IS durante el otoño de 2005 en la tabla *Offering*.

El ejemplo 9.28a no es muy útil porque es poco probable que todos los profesores hayan impartido todos los cursos. En lugar de ello, es más útil recuperar a los profesores que imparten un curso en cada semestre como lo muestra el Ejemplo 9.29. En vez de contar filas en cada

**EJEMPLO 9.28  
(Access)****Problema de división con una unión**

Haga una lista con el número de afiliación al Seguro Social y el nombre (nombre y apellido) de los profesores que imparten todos los cursos de sistemas de información en otoño de 2005.

```
SELECT Faculty.FacSSN, FacFirstName, FacLastName
  FROM Faculty, Offering
 WHERE Faculty.FacSSN = Offering.FacSSN
   AND OffTerm = 'FALL' AND CourseNo LIKE 'IS*'
   AND OffYear = 2005
GROUP BY Faculty.FacSSN, FacFirstName, FacLastName
HAVING COUNT(*) =
( SELECT COUNT(*) FROM Offering
  WHERE OffTerm = 'FALL' AND OffYear = 2005
    AND CourseNo LIKE 'IS*' )
```

FacSSN	FacFirstName	FacLastName
098-76-5432	LEONARD	VINCE

**EJEMPLO 9.28a Problema de división con una unión  
(Oracle)**

Haga una lista con el número de afiliación al Seguro Social y el nombre (nombre y apellido) de los profesores que impartieron todos los cursos de sistemas de información en otoño de 2005.

```
SELECT Faculty.FacSSN, FacFirstName, FacLastName
  FROM Faculty, Offering
 WHERE Faculty.FacSSN = Offering.FacSSN
   AND OffTerm = 'FALL' AND CourseNo LIKE 'IS%'
   AND OffYear = 2005
GROUP BY Faculty.FacSSN, FacFirstName, FacLastName
 HAVING COUNT(*) =
( SELECT COUNT(*) FROM Offering
  WHERE OffTerm = 'FALL' AND OffYear = 2005
    AND CourseNo LIKE 'IS%' )
```

grupo, cuente los valores únicos *CourseNo*. Este cambio es necesario porque *CourseNo* no es único en la tabla *Offering*. Puede haber varias filas con el mismo *CourseNo*, que corresponde a una situación en la que hay varias opciones para el mismo curso. La solución sólo se ejecuta en Oracle porque Access no ofrece soporte para la palabra clave DISTINCT en funciones conjuntas. El ejemplo 9.30 muestra una solución en Access que usa dos peticiones empaquetadas en cláusulas FROM. La segunda petición empaquetada ocurre dentro de la petición empaquetada en la cláusula HAVING. El apéndice 9.A muestra una alternativa a las peticiones empaquetadas en la cláusula FROM utilizando varios enunciados SELECT.

**EJEMPLO 9.29 Problema de división con DISTINCT dentro de COUNT  
(Oracle)**

Haga una lista con el número de afiliación al Seguro Social y el nombre (nombre y apellido) de los profesores que impartieron por lo menos una sección de todos los cursos de sistemas de información en el otoño de 2005.

```
SELECT Faculty.FacSSN, FacFirstName, FacLastName
  FROM Faculty, Offering
 WHERE Faculty.FacSSN = Offering.FacSSN
   AND OffTerm = 'FALL' AND CourseNo LIKE 'IS%'
   AND OffYear = 2005
GROUP BY Faculty.FacSSN, FacFirstName, FacLastName
 HAVING COUNT(DISTINCT CourseNo) =
( SELECT COUNT(DISTINCT CourseNo) FROM Offering
  WHERE OffTerm = 'FALL' AND OffYear = 2005
    AND CourseNo LIKE 'IS%' )
```

FacSSN	FacFirstName	FacLastName
098-76-5432	LEONARD	VINCE

**EJEMPLO 9.30  
(Access)****Problema de división usando la consulta empaquetada en FROM en lugar de la palabra clave DISTINCT dentro de la función COUNT**

Haga una lista con el número de seguridad social y el nombre (nombre y apellido) de los profesores que impartieron por lo menos una sección de todos los cursos de sistemas de información durante el otoño de 2005. El resultado es el mismo que el del ejemplo 9.29.

```

SELECT FacSSN, FacFirstName, FacLastName
  FROM
    (SELECT DISTINCT Faculty.FacSSN, FacFirstName, FacLastName,
      CourseNo
     FROM Faculty, Offering
    WHERE Faculty.FacSSN = Offering.FacSSN
      AND OffTerm = 'FALL' AND OffYear = 2005
      AND CourseNo LIKE 'IS*' )
  GROUP BY FacSSN, FacFirstName, FacLastName
 HAVING COUNT(*) =
  ( SELECT COUNT(*) FROM
    ( SELECT DISTINCT CourseNo
      FROM Offering
     WHERE OffTerm = 'FALL' AND OffYear = 2005
      AND CourseNo LIKE 'IS*' ) )

```

El ejemplo 9.31 es otra variación de la técnica empleada en el ejemplo 9.29. Es necesaria la palabra clave DISTINCT con el fin de no contar dos veces a los estudiantes que tomaron más de un curso con el mismo profesor. Observe que la palabra clave DISTINCT no es necesaria para la consulta empaquetada porque sólo se cuentan las filas de la tabla *Student*. El ejemplo 9.32 muestra una solución de Access que utiliza una consulta empaquetada con la cláusula FROM.

**EJEMPLO 9.31  
(Oracle)****Otro problema de división con DISTINCT dentro de COUNT**

Haga una lista con los profesores que impartieron todos los cursos de último grado en sistemas de información durante el otoño de 2005.

```

SELECT Faculty.FacSSN, FacFirstName, FacLastName
  FROM Faculty, Offering, Enrollment, Student
 WHERE Faculty.FacSSN = Offering.FacSSN
   AND OffTerm = 'FALL' AND CourseNo LIKE 'IS%'
   AND OffYear = 2005 AND StdClass = 'SR'
   AND Offering.OfferNo = Enrollment.OfferNo
   AND Student.StdSSN = Enrollment.StdSSN
  GROUP BY Faculty.FacSSN, FacFirstName, FacLastName
 HAVING COUNT(DISTINCT Student.StdSSN) =
  ( SELECT COUNT(*) FROM Student
    WHERE StdClass = 'SR' )

```

FacSSN	FacFirstName	FacLastName
098-76-5432	LEONARD	VINCE

**EJEMPLO 9.32  
(Access)****Otro problema de división con el uso de consultas empaquetadas en las cláusulas FROM, en lugar de la palabra clave DISTINCT dentro de la función COUNT**

Haga una lista con los profesores que impartieron todos los cursos del último grado en todos los cursos de sistemas de información durante el otoño de 2005. El resultado es idéntico al del ejemplo 9.31.

```
SELECT FacSSN, FacFirstName, FacLastName
  FROM
    ( SELECT DISTINCT Faculty.FacSSN, Faculty.FacFirstName,
      Faculty.FacLastName, Student.StdSSN
    FROM Faculty, Offering, Enrollment, Student
    WHERE Faculty.FacSSN = Offering.FacSSN
      AND OffTerm = 'FALL' AND CourseNo LIKE 'IS*'
      AND OffYear = 2005 AND StdClass = 'SR'
      AND Offering.OfferNo = Enrollment.OfferNo
      AND Student.StdSSN = Enrollment.StdSSN )
  GROUP BY FacSSN, FacFirstName, FacLastName
  HAVING COUNT(*) =
    ( SELECT COUNT(*) FROM Student
      WHERE StdClass = 'SR' )
```

## 9.4 Consideraciones de valor nulo

La última sección de este capítulo no comprende problemas difíciles de concordancia ni partes nuevas de la sentencia SELECT; en vez de ello, presenta la interpretación de los resultados de las consultas cuando las tablas contienen valores nulos. Hasta ahora hemos ignorado los efectos de estos valores, pues apenas en esta sección se simplificó su presentación. Como la mayoría de las bases de datos usan valores nulos, necesita entender sus efectos para conocer más a fondo la formulación de consultas.

Los valores nulos afectan las condiciones sencillas, que comprenden operadores de comparación, condiciones compuestas, que incluyen operadores lógicos, cálculos en conjunto y agrupaciones. Como verá, algunos de los efectos de los valores nulos son muy sutiles, por ello, un buen diseño de las tablas minimiza, aunque por lo regular no elimina, el uso de valores nulos. Los efectos nulos que se describen en esta sección se especifican en los estándares SQL-92, SQL:1999 y SQL:2003. En vista de que cada DBMS específico ofrece distintos resultados, tal vez necesite experimentar con el DBMS de su elección.

### 9.4.1 Efecto en condiciones simples

Las condiciones simples comprenden un operador de comparación, una columna o expresión de columna y una constante, una columna o una expresión de columna. Una condición sencilla da por resultado un valor nulo si cualquiera de las columnas (o expresión de columna) en una comparación es nula. Una fila influye en el resultado si la condición simple se evalúa como

**EJEMPLO 9.33****Condición simple con el uso de una columna con valores nulos**

Haga una lista de los clubes (tabla 9.11) con un presupuesto mayor de 200 dólares. Se omite el club con presupuesto nulo (C4) porque la condición se evalúa como un valor nulo.

```
SELECT *
  FROM Club
 WHERE CBudget > 200
```

ClubNo	CName	CPurpose	CBudget	CActual
C1	DELTA	SOCIAL	\$1 000.00	\$1 200.00
C2	BITS	ACADEMIC	\$500.00	\$350.00
C3	HELPS	SERVICE	\$300.00	\$330.00

verdadera para la fila. Las filas que se evalúan como falsas o nulas se descartan. El ejemplo 9.33 ilustra una condición simple que se evalúa como nula para una de las filas.

Puede darse el caso de un resultado más sutil cuando una condición simple comprende dos columnas y por lo menos una de ellas contiene valores nulos. Si ninguna columna contiene valores nulos, todas las filas estarán en el resultado de cualquiera de las condiciones simples o la opuesta (negación) de la condición simple. Por ejemplo, si  $<$  es el operador de una condición simple, la condición opuesta contiene  $\geq$  como su operador, suponiendo que las columnas permanecen en la misma posición. Si por lo menos una columna contiene valores nulos, algunas filas no aparecerán en el resultado de la condición simple ni de su negación. Para ser más precisos: las filas que contengan valores nuevos se van a ejecutar en ambos resultados, como lo muestran los ejemplos 9.34 y 9.35.

#### EJEMPLO 9.34 Condición simple de dos columnas

Enliste los clubes con un presupuesto mayor que los gastos actuales. Se omite el club con un presupuesto nulo (C4) porque la condición se evalúa como nula.

```
SELECT *
  FROM Club
 WHERE CBudget > CActual
```

ClubNo	CName	CPurpose	CBudget	CActual
C2	BITS	ACADEMIC	\$500.00	\$350.00

#### EJEMPLO 9.35 Condición opuesta del ejemplo 9.32

Enliste los clubes con un presupuesto menor o igual que los gastos actuales. Se omite el club con un presupuesto nulo (C4) porque la condición se evalúa como nula.

```
SELECT *
  FROM Club
 WHERE CBudget <= CActual
```

ClubNo	CName	CPurpose	CBudget	CActual
C1	DELTA	SOCIAL	\$1 000.00	\$1 200.00
C3	HELPS	SERVICE	\$300.00	\$330.00

#### 9.4.2 Efecto sobre las condiciones compuestas

Las condiciones compuestas comprenden una o más condiciones simples conectadas por los operadores lógicos o booleanos AND, OR y NOT. Al igual que las condiciones simples, las compuestas se evalúan como verdaderas, falsas o nulas. Se selecciona una fila si toda la condición compuesta en la cláusula WHERE se evalúa como verdadera.

Para evaluar el resultado de una condición compuesta, el estándar SQL:2003 utiliza tablas de verdad con tres valores. Una tabla de verdad muestra la forma en que las combinaciones

**TABLA 9.13**  
Tabla de verdad AND

<b>AND</b>	<b>Verdadero</b>	<b>Falso</b>	<b>Nulo</b>
<b>Verdadero</b>	Verdadero	Falso	Nulo
<b>Falso</b>	Falso	Falso	Falso
<b>Nulo</b>	Nulo	Falso	Nulo

**TABLA 9.14**  
Tabla de verdad OR

<b>OR</b>	<b>Verdadero</b>	<b>Falso</b>	<b>Nulo</b>
<b>Verdadero</b>	Verdadero	Verdadero	Verdadero
<b>Falso</b>	Verdadero	Falso	Nulo
<b>Nulo</b>	Verdadero	Nulo	Nulo

**TABLA 9.15**  
Tabla de verdad NOT

<b>NOT</b>	<b>Verdadero</b>	<b>Falso</b>	<b>Nulo</b>
	Falso	Verdadero	Nulo

de valores (verdadero, falso y nulo) se combinan con los operadores booleanos. Las tablas de verdad con tres valores definen una lógica de tres valores. Las tablas 9.13 a 9.15 ilustran las tablas de verdad para los operadores AND, OR y NOT. Las celdas internas de estas tablas son los valores resultantes. Por ejemplo, la primera celda interna (verdadera) en la tabla 9.13 resulta del operador AND aplicado a dos condiciones con valores verdaderos. Usted puede comprobar su comprensión con respecto a las tablas de verdad utilizando los ejemplos 9.36 y 9.37.

### EJEMPLO 9.36

#### Evaluación de una condición compuesta OR con un valor nulo

Enliste los clubes con un presupuesto menor o igual que sus gastos reales, o con un presupuesto real menor que 200 dólares. Se incluye el club con presupuesto nulo (C4) porque la condición se evalúa como verdadera.

```
SELECT *
  FROM Club
 WHERE CBudget <= CActual OR CActual < 200
```

ClubNo	CName	CPurpose	CBudget	CActual
C1	DELTA	SOCIAL	\$1 000.00	\$1 200.00
C3	HELPS	SERVICE	\$300.00	\$330.00
C4	SIGMA	SOCIAL		\$150.00

### EJEMPLO 9.37

#### Evaluación de una condición compuesta AND con un valor nulo

Enliste los clubes (tabla 9.11) con un presupuesto menor o igual que los gastos reales, o con gastos reales menores que 500 dólares. No se incluye el club con presupuesto nulo (C4) porque la primera condición se evalúa como nula.

```
SELECT *
  FROM Club
 WHERE CBudget <= CActual AND CActual < 500
```

ClubNo	CName	CPurpose	CBudget	CActual
C3	HELPS	SERVICE	\$300.00	\$330.00

### 9.4.3 Efecto sobre los cálculos conjuntos y la agrupación

Los valores nulos se ignoran en los cálculos conjuntos. Aunque esta sentencia parece sencilla, los resultados pueden ser sutiles. Para la función COUNT, COUNT(\*) regresa un valor diferente que COUNT(columna), si la columna contiene valores nulos. COUNT(\*) siempre regresa el número de filas. COUNT(columna) regresa el número de valores no nulos en la columna. El ejemplo 9.38 muestra la diferencia entre COUNT(\*) y COUNT(columna).

#### EJEMPLO 9.38

##### Función COUNT con valores nulos

Enliste el número de filas en la tabla *Club* y el número de valores en la columna *CBudget*.

```
SELECT COUNT(*) AS NumRows,
       COUNT(CBudget) AS NumBudgets
  FROM Club
```

NumRows	NumBudgets
4	3

Puede ocurrir un efecto todavía más sutil si se aplican las funciones SUM (suma) o AVG (promedio) a una columna con valores nulos. Sin tomar en cuenta los valores nulos, la siguiente ecuación es verdadera:  $SUM(Column1) + SUM(Column2) = SUM(Column1 + Column2)$ . La ecuación podría no ser verdadera con valores nulos en por lo menos una de las columnas, porque un cálculo que comprende un valor nulo produce otro valor nulo. Si Column1 tiene un valor nulo en una fila, la operación de suma en  $SUM(Column1 + Column2)$  produce un valor nulo para esa fila. Sin embargo, el valor de Column2 en la misma fila se cuenta en  $SUM(Column2)$ . El ejemplo 9.39 muestra este efecto sutil usando el operador menos en lugar del operador más.

#### EJEMPLO 9.39

##### Función SUM con valores nulos

Usando la tabla *Club*, haga una lista con la suma de los valores del presupuesto, la suma de los valores reales, la diferencia de ambas sumas y la suma de las diferencias (presupuesto – real). Las dos últimas columnas son diferentes porque hay un valor nulo en la columna *CBudget*. Los paréntesis encierran los valores negativos en el resultado.

```
SELECT SUM(CBudget) AS SumBudget,
       SUM(CActual) AS SumActual,
       SUM(CBudget)–SUM(CActual) AS SumDifference,
       SUM(CBudget–CActual) AS SumOfDifferences
  FROM Club
```

SumBudget	SumActual	SumDifference	SumOfDifferences
\$1 800.00	\$2 030.00	(\$230.00)	(\$80.00)

Los valores nulos también pueden afectar las operaciones agrupadas que se realizan en la cláusula GROUP BY. El estándar SQL estipula que todas las filas con valores nulos se deben agrupar. La columna agrupada muestra valores nulos en el resultado. En la base de datos de la universidad, este tipo de operación agrupada es útil para encontrar los cursos ofrecidos que no tienen profesores asignados, como lo muestra el ejemplo 9.40.

**EJEMPLO 9.40****Agrupación en una columna con valores nulos**

Por cada número de seguridad social de los profesores en la tabla *Offering*, haga una lista del número de cursos. En Microsoft Access y Oracle, la fila *Offering* con un valor nulo *FacSSN* se despliega como un espacio en blanco. En Access, la fila nula aparece antes que las filas no nulas, como se muestra a continuación. En Oracle, la fila nula aparece después de las filas no nulas.

```
SELECT FacSSN, COUNT(*) AS NumRows
  FROM Offering
 GROUP BY FacSSN
```

FacSSN	NumRows
	2
098-76-5432	3
543-21-0987	1
654-32-1098	2
765-43-2109	2
876-54-3210	1
987-65-4321	2

## Reflexión final

El capítulo 9 expuso las posibilidades de formulación de consultas avanzadas poniendo énfasis en la concordancia compleja entre las tablas y un subconjunto más extenso de SQL. Los problemas de concordancia compleja incluyen el enlace externo con sus variaciones (un lado y completa), así como problemas que requieren de operadores de diferencia y división de álgebra relacional. Además de conocer nuevos tipos de problemas y nuevas partes de la sentencia SELECT, este capítulo explicó los efectos sutiles de los valores nulos con el fin de ofrecerle un conocimiento profundo de la formulación de consultas.

Se han abarcado dos partes nuevas de la sentencia SELECT. Las palabras clave LEFT, RIGHT y FULL como parte del estilo de operador de enlace soportan las operaciones de enlace externo. Las consultas empaquetadas son una consulta dentro de otra. Para entender el efecto de una consulta empaquetada, busque las tablas utilizadas tanto en una consulta externa como interna. Si no existen tablas comunes, la consulta empaquetada se ejecuta una vez (consulta empaquetada tipo I); de lo contrario, la consulta empaquetada se ejecuta una vez por cada fila de la consulta externa (consulta empaquetada tipo II). Las consultas empaquetadas tipo I casi siempre se utilizan para formular enlace como parte de las sentencias SELECT y DELETE. Las consultas empaquetadas tipo I con el operador NOT IN y las consultas empaquetadas tipo II con el operador NOT EXISTS son útiles para problemas que comprenden el operador de diferencia. Las consultas empaquetadas tipo I con la cláusula HAVING son útiles para problemas que comprenden el operador de división.

Aunque las habilidades para realizar consultas avanzadas no se requieren con la misma frecuencia que aquéllas para realizar las consultas fundamentales que cubrimos en el capítulo 4, es importante tenerlas en consideración para cuando se requieran. Tal vez se dé cuenta de que obtiene una ventaja competitiva al dominar también las habilidades avanzadas.

Los capítulos 4 y 9 ofrecen importantes habilidades para la formulación de consultas y las partes de la sentencia SELECT de SQL. A pesar de esta importante cobertura, todavía queda mucho que aprender. Existen problemas de concordancia todavía más complejos y otras partes de la sentencia SELECT que no hemos descrito. Le recomendamos ampliar sus habilidades consultando las referencias que se citan al final del capítulo. Además, aún no ha aprendido a aplicar sus habilidades para la formulación de consultas en la construcción de aplicaciones. El capítulo 10 desarrolla sus habilidades en la construcción de aplicaciones con vistas, mientras que el capítulo 11 aplica estas habilidades en el uso de procedimientos almacenados y disparadores (*triggers*).

## Revisión de conceptos

- Formulación de enlaces externos de un lado con Access y Oracle (9i y posterior).
 

```
SELECT OfferNo, CourseNo, Offering.FacSSN, Faculty.FacSSN,
         FacFirstName, FacLastName
    FROM Offering LEFT JOIN Faculty
      ON Offering.FacSSN = Faculty.FacSSN
     WHERE CourseNo = 'IS480'
```
- Formulación de enlaces externos completos con el uso de la palabra clave FULL JOIN (SQL:2003 y Oracle 9i y posterior).
 

```
SELECT FacSSN, FacFirstName, FacLastName, FacSalary,
         StdSSN, StdFirstName, StdLastName, StdGPA
    FROM Faculty FULL JOIN Student
      ON Student.StdSSN = Faculty.FacSSN
```
- Formulación de enlaces externos completos combinando dos enlaces externos de un lado en Access.
 

```
SELECT FacSSN, FacFirstName, FacLastName, FacSalary,
         StdSSN, StdFirstName, StdLastName, StdGPA
    FROM Faculty RIGHT JOIN Student
      ON Student.StdSSN = Faculty.FacSSN
        UNION
SELECT FacSSN, FacFirstName, FacLastName, FacSalary,
         StdSSN, StdFirstName, StdLastName, StdGPA
    FROM Faculty LEFT JOIN Student
      ON Student.StdSSN = Faculty.FacSSN
```
- Combinación de enlaces internos y externos (Access y Oracle 9i y posterior).
 

```
SELECT OfferNo, Offering.CourseNo, OffTerm, CrsDesc,
         Faculty.FacSSN, FacFirstName, FacLastName
    FROM ( Faculty RIGHT JOIN Offering
      ON Offering.FacSSN = Faculty.FacSSN )
      INNER JOIN Course
        ON Course.CourseNo = Offering.CourseNo
     WHERE OffYear = 2006
```
- Comprender que las condiciones en las cláusulas WHERE o HAVING pueden usar las sentencias SELECT, además de valores de escalares (individuales).
- Identificar las consultas empaquetadas tipo I por la palabra clave IN y la falta de referencia a una tabla utilizada en una consulta externa.
- Uso de una consulta empaquetada tipo I para formular un enlace.
 

```
SELECT DISTINCT StdSSN, StdFirstName, StdLastName,
         StdMajor
    FROM Student
   WHERE Student.StdSSN IN
      ( SELECT StdSSN FROM Enrollment
        WHERE EnrGrade >= 3.5 )
```
- Uso de una consulta empaquetada tipo I dentro de una sentencia DELETE para probar las condiciones en una tabla relacionada.

```

DELETE FROM Offering
WHERE Offering.FacSSN IN
(   SELECT FacSSN FROM Faculty
    WHERE FacFirstName = 'LEONARD'
        AND FacLastName = 'VINCE'  )

```

- No usar una consulta empaquetada tipo I para un enlace cuando se necesita una columna de la consulta empaquetada en el resultado final de la consulta.
- Identificación de sentencias de problemas que comprenden el operador de diferencia: las palabras *not* u *only* relacionando dos nombres en una oración.
- Formulaciones limitadas de SQL para problemas de diferencia: consultas empaquetadas tipo I con el operador NOT IN, enlace externo de un lado con una condición IS NULL y operación de diferencia usando las palabras clave EXCEPT o MINUS.
- Uso de una consulta empaquetada tipo I con el operador NOT IN para problemas de diferencia que comprenden una comparación de columnas.

```

SELECT FacSSN, FacFirstName, FacLastName, FacDept,
      FacSalary
FROM Faculty
WHERE FacSSN NOT IN
(   SELECT StdSSN FROM Student  )

```

- Identificación de consultas empaquetadas tipo II por una referencia a la tabla utilizada en una consulta externa.
- Uso de consultas empaquetadas tipo II con el operador NOT EXISTS para problemas de diferencia complejos.

```

SELECT FacSSN, FacFirstName, FacLastName, FacDept,
      FacSalary
FROM Faculty
WHERE NOT EXISTS
(   SELECT * FROM Student
    WHERE Student.StdSSN = Faculty.FacSSN  )

```

- Uso de una consulta empaquetada en la cláusula FROM para calcular conjuntos empaquetados o conjuntos para más de una agrupación.

```

SELECT T.CourseNo, T.CrsDesc, COUNT(*) AS NumOfferings,
      Avg(T.EnrollCount) AS AvgEnroll
FROM
(   SELECT Course.CourseNo, CrsDesc,
          Offering.OfferNo, COUNT(*) AS EnrollCount
      FROM Offering, Enrollment, Course
      WHERE Offering.OfferNo = Enrollment.OfferNo
          AND Course.CourseNo = Offering.CourseNo
      GROUP BY Course.CourseNo, CrsDesc, Offering.OfferNo  ) T
GROUP BY T.CourseNo, T.CrsDesc

```

- Identificación de sentencias de problemas que comprenden el operador de división: las palabras *every* y *all* conectando distintas partes de una oración.
- Uso del método de contar para formular problemas de división.

```

SELECT StdNo
FROM StdClub
GROUP BY StdNo
HAVING COUNT(*) = (   SELECT COUNT(*) FROM Club  )

```

- Evaluación de una condición sencilla que contiene un valor nulo en una expresión de columna.
- Uso de la lógica de tres valores y tablas de verdad para evaluar condiciones compuestas con valores nulos.
- Entender el resultado de los cálculos de conjunto con valores nulos.
- Entender el resultado de la agrupación en una columna de valores nulos.

## Preguntas

1. Explique una situación en la que es útil un enlace externo de un lado.
2. Explique una situación en la que es útil un enlace externo completo.
3. ¿Cómo interpreta el significado de las palabras clave LEFT y RIGHT JOIN en la cláusula FROM?
4. ¿Cuál es la interpretación de la palabra clave FULL JOIN en la cláusula FROM?
5. ¿Cómo lleva a cabo un enlace externo completo en las implementaciones SQL (como Microsoft Access y Oracle 8i) que no ofrecen soporte para la palabra clave FULL JOIN?
6. ¿Qué es una consulta empaquetada?
7. ¿Cuál es la característica distintiva de la apariencia de las consultas empaquetadas tipo I?
8. ¿Cuál es la característica distintiva de la apariencia de las consultas empaquetadas tipo II?
9. ¿Cuántas veces se ejecuta una consulta empaquetada tipo I como parte de una consulta externa?
10. ¿En qué aspectos una consulta empaquetada tipo I es como un procedimiento en un programa?
11. ¿Cuántas veces se ejecuta una consulta empaquetada tipo II como parte de una consulta externa?
12. ¿En qué aspectos una consulta empaquetada tipo II es como un ciclo empaquetado en un programa?
13. ¿Cuál es el significado del operador de comparación IN?
14. ¿Cuál es el significado del operador de comparación EXISTS?
15. ¿Cuál es el significado del operador de comparación NOT EXISTS?
16. ¿En qué momento puede dejar de usarse una consulta empaquetada tipo I para realizar un enlace?
17. ¿Por qué una consulta empaquetada tipo I es un buen método de enlace cuando necesita un enlace en una sentencia DELETE?
18. ¿Por qué SQL:2003 permite consultas empaquetadas en la cláusula FROM?
19. Identifique dos situaciones en las que son necesarias las consultas empaquetadas en la cláusula FROM.
20. ¿Cómo detecta que un problema comprende una operación de división?
21. Explique el método “contar” para formular problemas de división.
22. ¿Por qué a veces es necesario utilizar la palabra clave DISTINCT dentro de la función COUNT para problemas de división?
23. ¿Cuál es el resultado de una condición sencilla cuando una expresión de columna en la condición se evalúa como nula?
24. ¿Qué es una tabla de verdad?
25. ¿Cuántos valores tienen las tablas de verdad en el estándar SQL:2003?
26. ¿Cómo utiliza las tablas de verdad para evaluar las condiciones compuestas?
27. ¿De qué manera los valores nulos afectan los cálculos en conjunto?
28. Explique por qué la siguiente ecuación podría no ser verdadera si Column1 o Column2 contienen valores nulos:  $SUM(Column1) - SUM(Column2) = SUM(Column1 - Column2)$ .
29. ¿Cómo se manejan los valores nulos en una columna agrupada?
30. En Access, ¿cómo compensa la falta de la palabra clave DISTINCT dentro de la función COUNT?
31. ¿Cuándo puede utilizar una consulta empaquetada tipo I con el operador NOT IN para formular una operación de diferencia en SQL?

32. ¿Cuándo puede usar un enlace externo de un lado con una condición IS NULL para formular una operación de diferencia en SQL?
33. ¿Cuándo puede utilizar una operación MINUS en SQL para formular una operación de diferencia en SQL?
34. ¿Cuál es la forma más general de formular operaciones de diferencia en sentencias SQL?

## Problemas

Los problemas usan las tablas de la base de datos Order Entry (captura de pedidos) en la sección de problemas del capítulo 4. Al formular los problemas, recuerde que la llave foránea *EmpNO* en la tabla *OrderTbl* permite valores nulos. Un pedido no debe tener un empleado asociado si se toma a través de Internet.

1. Utilizando una consulta empaquetada tipo I haga una lista con el número de cliente, nombre (nombre y apellido) y ciudad de cada cliente que tenga un saldo mayor que 150 dólares y haya hecho un pedido en febrero de 2007.
2. Utilizando una consulta empaquetada tipo II haga una lista con el número de cliente, nombre (nombre y apellido) y ciudad de cada cliente que tenga un saldo mayor que 150 dólares y haya hecho un pedido en febrero de 2007.
3. Usando dos consultas empaquetadas tipo I haga una lista con el número de producto, nombre y precio de los productos con un precio mayor que 150 dólares y que se hayan pedido el 23 de enero de 2007.
4. Usando dos consultas empaquetadas tipo II y otro estilo de enlace, haga una lista con el número de producto, nombre y precio de los productos con un precio mayor que 150 dólares y que se pidieron en enero de 2007 por clientes con un saldo mayor que 400 dólares.
5. Haga una lista con el número y fecha de pedido, nombre (nombre y apellido) y número de empleado de los pedidos hechos el 23 de enero de 2007. Incluya el pedido aun cuando no haya un empleado asociado.
6. Haga una lista con el número y fecha de pedido, número de empleado, nombre (nombre y apellido) del empleado y número y nombre de cliente de los pedidos hechos el 23 de enero de 2007. Incluya el pedido aun cuando no haya un empleado asociado.
7. Haga una lista de todas las personas en la base de datos. La tabla resultante deberá tener todas las columnas de las tablas *Customer* y *Employee*. Compare los nombres y apellidos en las tablas *Customer* y *Employee*. Si un cliente no concuerda con ningún empleado, las columnas que pertenecen a la tabla *Employee* quedarán en blanco. De modo similar, en el caso en que un empleado no concuerde con ningún cliente, las columnas pertenecientes a la tabla *Customer* quedarán en blanco.
8. Para cada producto Ink Jet pedido en enero de 2007, haga una lista del número de pedido, fecha del pedido, número de cliente, nombre del cliente (nombre y apellido), número de empleado (si lo hay), nombre de empleado (nombre y apellido), cantidad pedida, número de producto y nombre de producto. Incluya los productos que contengan Ink Jet en el campo nombre de producto. Incluya tanto los pedidos por Internet (sin empleado) como por teléfono (tomados por un empleado).
9. Usando una consulta empaquetada tipo II haga una lista con el número y nombre de los clientes de Colorado que no hicieron ningún pedido en febrero de 2007.
10. Repita el problema 9 usando una consulta empaquetada tipo I con una condición NOT IN en lugar de una consulta empaquetada. Si no puede formular el problema de esta manera, dé una explicación indicando la razón.
11. Repita el problema 9 utilizando la palabra clave MINUS. Observe que Access no ofrece soporte para la palabra clave MINUS. Si no puede formular el problema de esta manera, dé una explicación indicando la razón.
12. Repita el problema 9 usando un enlace externo de un lado y una condición IS NULL. Si no puede formular el problema de esta manera, dé una explicación indicando la razón.
13. Usando una consulta empaquetada tipo II haga una lista con el número de empleado, nombre y apellido de los empleados con el código postal (720) que no han recibido pedidos. Un empleado se encuentra en el código postal (720) si su número de teléfono contiene la hilera (720) al principio del valor de la columna.
14. Repita el problema 13 usando una consulta empaquetada tipo I con una condición NOT IN en lugar de una consulta empaquetada. Si no puede formular el problema de esta manera, dé una explicación

- indicando la razón. (*Consejo:* Debe pensar con detenimiento el efecto de los valores nulos en la columna *OrderTbl.EmpNo.*)
15. Repita el problema 9 usando un enlace externo de un lado y una condición IS NULL. Si no puede formular el problema de esta manera, dé una explicación indicando la razón.
  16. Repita el problema 9 usando la palabra clave MINUS. Recuerde que Access no ofrece soporte para la palabra clave MINUS. Si no puede formular el problema de esta manera, dé una explicación indicando la razón.
  17. Haga una lista con el número de pedido y la fecha de los mismos que contienen un solo producto con las palabras Ink Jet en la descripción.
  18. Haga una lista con el número de cliente y el nombre (nombre y apellido) de quienes pidieron productos fabricados por Connex. Incluya sólo los clientes que pidieron por lo menos un producto fabricado por Connex. Elimine del resultado las filas que se repitan.
  19. Haga una lista con el número de pedido y la fecha de los mismos que contienen todos los productos con las palabras Ink Jet en la descripción.
  20. Haga una lista con el número y nombre de los productos contenidos en los pedidos hechos entre el 7 y el 9 de enero de 2007.
  21. Haga una lista con el número y nombre (nombre y apellido) de los clientes que pidieron productos fabricados por ColorMeg, Inc. en enero de 2007.
  22. Usando una consulta empaquetada tipo I, elimine los pedidos hechos por la cliente Betty Wise en enero de 2007. La acción CASCADE DELETE (eliminación en cascada) borrará todas las filas en la tabla *OrdLine*.
  23. Usando una consulta empaquetada tipo I, elimine los pedidos hechos por los clientes de Colorado y que recibió Landi Santos en enero de 2007. La acción CASCADE DELETE va a borrar las filas relacionadas en la tabla *OrdLine*.
  24. Haga una lista con el número y fecha de los pedidos en los que cualquier parte de la dirección de envío (calle, ciudad, estado y código postal) es distinta a la dirección del cliente.
  25. Haga una lista con el número y nombre (nombre y apellido) de los empleados que recibieron pedidos de cualquier cliente de Seattle durante enero de 2007.
  26. Para los clientes de Colorado, calcule el monto promedio de sus pedidos. El monto promedio de los pedidos de un cliente es la suma del precio (cantidad pedida por precio del producto) de cada pedido dividida entre la cantidad de pedidos. El resultado debe incluir el número y el apellido del cliente, y el monto promedio de los pedidos.
  27. Para los clientes de Colorado, calcule el monto promedio de sus pedidos y el número de pedidos realizados. El resultado debe incluir el número y el apellido del cliente, el monto promedio de los pedidos y el número de pedidos realizados. En Access es muy difícil formular este problema.
  28. Para los clientes de Colorado, calcule el número de productos únicos ordenados. Si un producto se compra en varios pedidos, sólo se debe contar una vez. El resultado debe incluir el número y el apellido del cliente, y el número de productos únicos ordenados.
  29. Para cada empleado con una comisión menor que 0.04, calcule el número de pedidos tomados y el número promedio de productos por pedido. El resultado debe incluir el número y el apellido del empleado, el número de pedidos recibidos y el número promedio de productos por pedido. En Access es muy difícil formular este problema como una sola sentencia SELECT.
  30. Calcule el número de clientes únicos que pidieron cada producto de Connex en enero de 2007. El resultado debe incluir número y nombre del producto, así como el número de clientes únicos.

## Problemas de valor nulo

Los siguientes problemas se basan en las tablas *Product* (producto) y *Employee* (empleado) de la base de datos Order Entry. Para su mayor comodidad hemos reproducido nuevamente las tablas. La columna *ProdNextShipDate* contiene la fecha de envío esperada para el producto. Si el valor es nulo, no se ha acordado un nuevo envío. No se programa un envío por diversas razones, como la gran cantidad de envíos pendientes o el hecho de que el fabricante no tenga el producto disponible. En la tabla *Employee*, la tasa de comisión puede ser nula, lo que indica que no se ha asignado una tasa de comisión. Un valor nulo para *SupEmpNo* indica que el empleado no tiene un superior.

### Producto

ProdNo	ProdName	ProdMfg	ProdQOH	ProdPrice	ProdNextShipDate
P0036566	17-inch Color Monitor	ColorMeg, Inc.	12	\$169.00	2/20/2007
P0036577	19-inch Color Monitor	ColorMeg, Inc.	10	\$319.00	2/20/2007
P1114590	R3000 Color Laser Printer	Connex	5	\$699.00	1/22/2007
P1412138	10-Foot Printer Cable	Ethlite	100	\$12.00	
P1445671	8-Outlet Surge Protector	Intersafe	33	\$14.99	
P1556678	CVP Ink Jet Color Printer	Connex	8	\$99.00	1/22/2007
P3455443	Color Ink Jet Cartridge	Connex	24	\$38.00	1/22/2007
P4200344	36-Bit Color Scanner	UV Components	16	\$199.99	1/29/2007
P6677900	Black Ink Jet Cartridge	Connex	44	\$25.69	
P9995676	Battery Back-up System	Cybercx	12	\$89.00	2/1/2007

### Empleado

EmpNo	EmpFirstName	EmpLastName	EmpPhone	EmpEmail	SupEmpNo	EmpCommRate
E1329594	Landi	Santos	(303) 789-1234	LSantos@bigco.com	E8843211	0.02
E8544399	Joe	Jenkins	(303) 221-9875	JJenkins@bigco.com	E8843211	0.02
E8843211	Amy	Tang	(303) 556-4321	ATang@bigco.com	E9884325	0.04
E9345771	Colin	White	(303) 221-4453	CWhite@bigco.com	E9884325	0.04
E9884325	Thomas	Johnson	(303) 556-9987	TJohnson@bigco.com		0.05
E9954302	Mary	Hill	(303) 556-9871	MHill@bigco.com	E8843211	0.02
E9973110	Theresa	Beck	(720) 320-2234	TBeck@bigco.com	E9884325	

1. Identifique las filas del resultado para la siguiente sentencia SELECT. Se muestran las versiones de la sentencia en Access y Oracle.

Access:

```
SELECT *
FROM Product
WHERE ProdNextShipDate = #1/22/2007#
```

Oracle:

```
SELECT *
FROM Product
WHERE ProdNextShipDate = '22-Jan-2007';
```

2. Identifique las filas del resultado para la siguiente sentencia SELECT:

Access:

```
SELECT *
FROM Product
WHERE ProdNextShipDate = #1/22/2007#
AND ProdPrice < 100
```

Oracle:

```
SELECT *
FROM Product
WHERE ProdNextShipDate = '22-Jan-2007'
AND ProdPrice < 100;
```

3. Identifique las filas del resultado para la siguiente sentencia SELECT:

Access:

```
SELECT *
  FROM Product
 WHERE ProdNextShipDate = #1/22/2007#
   OR ProdPrice < 100
```

Oracle:

```
SELECT *
  FROM Product
 WHERE ProdNextShipDate = '22-Jan-2007'
   OR ProdPrice < 100;
```

4. Determine el resultado para la siguiente sentencia SELECT:

```
SELECT COUNT(*) AS NumRows,
       COUNT(ProdNextShipDate) AS NumShipDates
  FROM Product
```

5. Determine el resultado para la siguiente sentencia SELECT:

```
SELECT ProdNextShipDate, COUNT(*) AS NumRows
  FROM Product
 GROUP BY ProdNextShipDate
```

6. Determine el resultado para la siguiente sentencia SELECT:

```
SELECT ProdMfg, ProdNextShipDate, COUNT(*) AS NumRows
  FROM Product
 GROUP BY ProdMfg, ProdNextShipDate
```

7. Determine el resultado para la siguiente sentencia SELECT:

```
SELECT ProdNextShipDate, ProdMfg, COUNT(*) AS NumRows
  FROM Product
 GROUP BY ProdNextShipDate, ProdMfg
```

8. Identifique las filas del resultado para la siguiente sentencia SELECT:

```
SELECT EmpFirstName, EmpLastName
  FROM Employee
 WHERE EmpCommRate > 0.02
```

9. Determine el resultado de la siguiente sentencia SELECT:

```
SELECT SupEmpNo, AvG(EmpCommRate) AS AvgCommRate
  FROM Employee
 GROUP BY SupEmpNo
```

10. Determine el resultado de la siguiente sentencia SELECT. La sentencia calcula la tasa de comisión promedio de los empleados subordinados. El resultado incluye el número de empleado, nombre y apellido del supervisor, así como la cantidad promedio de comisión para los subordinados.

```
SELECT Emp.SupEmpNo, Sup.EmpFirstName, Sup.EmpLastName,
       AvG(Emp.EmpCommRate) AS AvgCommRate
  FROM Employee Emp, Employee Sup
 WHERE Emp.SupEmpNo = Sup.EmpNo
 GROUP BY Emp.SupEmpNo, Sup.EmpFirstName, Sup.EmpLastName
```

11. Utilice sus conocimientos sobre la evaluación del valor nulo para explicar por qué estos dos sentencias SQL generan distintos resultados para la base de datos Order Entry. Recuerde que no se permiten valores nulos para *OrderTbl.EmpNo*.

```

SELECT EmpNo, EmpLastName, EmpFirstName
  FROM Employee
 WHERE EmpNo NOT IN
 (  SELECT EmpNo FROM OrderTbl WHERE EmpNo IS NOT NULL  )

SELECT EmpNo, EmpLastName, EmpFirstName
  FROM Employee
 WHERE EmpNo NOT IN
 (  SELECT EmpNo FROM OrderTbl  )

```

## Referencias para ampliar su estudio

La mayor parte de los libros de texto para estudiantes de administración de empresas no cubren la formulación de consultas o el SQL con tanto detalle como se hace aquí. Para un aprendizaje avanzado de SQL más allá de lo aquí expuesto, deberá consultar una reseña de libros sobre SQL en [www.ocelot.ca/books.htm](http://www.ocelot.ca/books.htm). Para las nuevas características de SQL:1999, consulte a Melton y Simon (2001). Groff y Weinberg (1999) cubren las diversas anotaciones para enlaces externos disponibles en los DBMS comerciales. Los sitios *DBAZone* ([www.dbazine.com](http://www.dbazine.com)) y *DevX.com Database Zone* ([www.devx.com](http://www.devx.com)) ofrecen muchos consejos prácticos sobre la formulación de consultas en SQL. Para consejos específicos sobre productos SQL, el sitio *Advisor.com* ([www.advisor.com](http://www.advisor.com)) ofrece diarios técnicos para Microsoft SQL Server y Microsoft Access. La documentación de Oracle se encuentra en el sitio *Oracle Technet* ([www.oracle.com/technology](http://www.oracle.com/technology)).

## Apéndice 9.A

### Uso de sentencias múltiples en Microsoft Access

En Microsoft Access puede usar varias sentencias SELECT en lugar de consultas empaquetadas con la cláusula FROM. En algunos casos, el uso de varias sentencias ofrece una formulación más sencilla que utilizar consultas empaquetadas con la cláusula FROM. Por ejemplo, en lugar de usar DISTINCT dentro de COUNT, como en el ejemplo 9.29, puede utilizar una consulta guardada con la palabra clave DISTINCT después de SELECT. En el ejemplo 9A.1, la primera consulta guardada (Temp9A-1) encuentra las únicas combinaciones de nombre de profesor y número de curso. Note que el uso de la palabra clave DISTINCT elimina las duplicaciones. La segunda consulta guardada (Temp9A-2) encuentra los cursos únicos en la tabla *Offering*. La última consulta combina las dos consultas guardadas. Recuerde que puede usar las consultas guardadas de manera similar a como se utilizan las tablas; sólo tiene que usar el nombre de la consulta guardada en la cláusula FROM.

#### EJEMPLO 9A.1

#### Uso de consultas guardadas en lugar de consultas empaquetadas con la cláusula FROM

Haga una lista con los profesores que imparten por lo menos una sección de todos los cursos de sistemas de la información durante el otoño de 2005. El resultado es idéntico al del ejemplo 9.29.

## Temp9A-1

```
SELECT DISTINCT Faculty.FacSSN, FacFirstName, FacLastName, CourseNo
  FROM Faculty, Offering
 WHERE Faculty.FacSSN = Offering.FacSSN
   AND OffTerm = 'FALL' AND OffYear = 2005
   AND CourseNo LIKE 'IS*' 
```

## Temp9A-2

```
SELECT DISTINCT CourseNo
  FROM Offering
 WHERE OffTerm = 'FALL' AND OffYear = 2005
   AND CourseNo LIKE 'IS*' 
```

```
SELECT FacSSN, FacFirstName, FacLastName
  FROM [Temp9A-1]
 GROUP BY FacSSN, FacFirstName, FacLastName
 HAVING COUNT(*) = ( SELECT COUNT(*) FROM [Temp9A-2] ) 
```

## Apéndice 9.B

### Resumen de la sintaxis de SQL:2003

Este apéndice resume la sintaxis de SQL:2003 para las sentencias SELECT empaquetados (subpeticiones), así como las operaciones de enlace externo que presentamos en el capítulo 9. Para la sintaxis de otras variaciones de la sentencia SELECT empaquetada y las operaciones de enlace externo que aparecen en el capítulo 9, consulte un libro sobre SQL:1999 o SQL:2003. Las sentencias SELECT empaquetadas se pueden utilizar en las cláusulas FROM y WHERE de las sentencias SELECT, UPDATE y DELETE. Las convenciones usadas en la sintaxis son idénticas a las utilizadas al final del capítulo 3.

### Sintaxis extendida para consultas empaquetadas en la cláusula FROM

```
<Table-Specification>:
{ <Simple-Table> | -- defined in Chapter 4
  <Join-Operation> | -- defined in Chapter 4
  <Simple-Select> [ [ AS ] AliasName ] }
-- <Simple-Select> is defined in Chapter 4 
```

## Sintaxis extendida para las condiciones de fila

```

<Row-Condition>:
{ <Simple-Condition> | -- defined in Chapter 4
  <Compound-Condition> | -- defined in Chapter 4
  <Exists-Condition> |
  <Element-Condition> }

<Exists-Condition>: [NOT] EXISTS <Simple-Select>

<Simple-Select>: -- defined in Chapter 4

<Element-Condition>:
<Scalar-Expression> <Element-Operator>(<Simple-Select> )

<Element-Operator>:
{ = | < | > | >= | <= | <> | [ NOT ] IN }

<Scalar-Expression>: -- defined in Chapter 4

```

## Sintaxis extendida para las condiciones de grupo

```

<Simple-Group-Condition>: -- Last choice is new
{ <Column-Expression> ComparisonOperator <Column-Expression> |
  <Column-Expression> [NOT] IN ( Constant* ) |
  <Column-Expression> BETWEEN <Column-Expression>
    AND <Column-Expression> |
  <Column-Expression> IS [NOT] NULL |
  ColumnName [NOT] LIKE StringPattern |
  <Exists-Condition> |
  <Column-Expression> <Element-Operator> <Simple-Select> }

<Column-Expression>: -- defined in Chapter 4

```

## Sintaxis extendida para operaciones de enlace externo

```

<Join-Operation>:
{ <Simple-Table> <Join-Operator> <Simple-Table>
  ON <Join-Condition> |
  { <Simple-Table> | <Join-Operation> } <Join-Operator>
    { <Simple-Table> | <Join-Operation> }
  ON <Join-Condition> |
  ( <Join-Operation> ) }

```

```
<Join-Operator>:
{ [ INNER ] JOIN |
  LEFT [ OUTER ] JOIN |
  RIGHT [ OUTER ] JOIN |
  FULL [ OUTER ] JOIN }
```

## Apéndice 9.C

### Anotación de Oracle 8i para enlaces externos

Las versiones de Oracle anteriores a la 9i utilizaban una extensión patentada para enlaces externos de un lado. Para expresar un enlace externo de un lado en Oracle 8i SQL necesita usar la notación (+) como parte de la condición de enlace en la cláusula WHERE. Debe colocar la notación (+) inmediatamente después de la columna de enlace en la tabla nula; es decir, la tabla con los valores nulos en el resultado. En contraste, las palabras clave LEFT y RIGHT de SQL:2003 se colocan después de la tabla en la que las filas de no concordancia se conservan en el resultado. Las formulaciones de los ejemplos 9.1, 9.2, 9.3, 9.4 y 9.5 en Oracle 8i demuestran la notación (+).

#### EJEMPLO 9.1 (Oracle 8i)

##### **Enlace externo de un lado con símbolo de enlace externo al lado derecho de la condición de enlace**

La notación (+) se coloca después de la columna *Faculty.FacSSN* en la condición de enlace, porque *Faculty* es la tabla nula en el resultado.

```
SELECT OfferNo, CourseNo, Offering.FacSSN, Faculty.FacSSN,
       FacFirstName, FacLastName
  FROM Faculty, Offering
 WHERE Offering.FacSSN = Faculty.FacSSN (+)
   AND CourseNo LIKE 'IS%'
```

#### EJEMPLO 9.2 (Oracle 8i)

##### **Símbolo de enlace externo en enlace externo de un lado al lado izquierdo de una condición de enlace**

La notación (+) se coloca después de la columna *Faculty.FacSSN* en la condición de enlace, porque la tabla *Faculty* es nula en el resultado.

```
SELECT OfferNo, CourseNo, Offering.FacSSN, Faculty.FacSSN,
       FacFirstName, FacLastName
  FROM Faculty, Offering
 WHERE Faculty.FacSSN (+) = Offering.FacSSN
   AND CourseNo LIKE 'IS%'
```

**EJEMPLO 9.3  
(Oracle 8i)****Enlace externo completo con el uso de un enlace de dos enlaces externos de un lado**

Combine las tablas *Faculty* y *Student* utilizando un enlace externo completo. Haga una lista con el número de seguridad social, nombre (nombre y apellido), salario (sólo profesores) y GPA (sólo estudiantes) en el resultado.

```
SELECT FacSSN, FacFirstName, FacLastName, FacSalary,
       StdSSN, StdFirstName, StdLastName, StdGPA
  FROM Faculty, Student
 WHERE Student.StdSSN = Faculty.FacSSN (+)
      UNION
SELECT FacSSN, FacFirstName, FacLastName, FacSalary,
       StdSSN, StdFirstName, StdLastName, StdGPA
  FROM Faculty, Student
 WHERE Student.StdSSN (+) = Faculty.FacSSN
```

**EJEMPLO 9.4  
(Oracle 8i)****Combinación de un enlace externo de un lado y un enlace interno**

Combine las columnas de las tablas *Faculty*, *Offering* y *Course* para los cursos de IS ofrecidos durante 2006. Incluya una fila en el resultado aun cuando no haya profesor asignado.

```
SELECT OfferNo, Offering.CourseNo, OffTerm, CrsDesc,
       Faculty.FacSSN, FacFirstName, FacLastName
  FROM Faculty, Offering, Course
 WHERE Offering.FacSSN = Faculty.FacSSN (+)
    AND Course.CourseNo = Offering.CourseNo
    AND Course.CourseNo LIKE 'IS%' AND OffYear = 2006
```

**EJEMPLO 9.5  
(Oracle 8i)****Combinación de un enlace externo de un lado y dos enlaces internos**

Haga una lista con las filas de la tabla *Offering* en las que hay por lo menos un estudiante inscrito, además de los requisitos del ejemplo 9.6. Elimine las filas repetidas cuando haya más de un estudiante inscrito en un curso.

```
SELECT DISTINCT Offering.OfferNo, Offering.CourseNo,
       OffTerm, CrsDesc, Faculty.FacSSN, FacFirstName,
       FacLastName
  FROM Faculty, Offering, Course, Enrollment
 WHERE Offering.FacSSN = Faculty.FacSSN (+)
    AND Course.CourseNo = Offering.CourseNo
    AND Offering.OfferNo = Enrollment.OfferNo
    AND Course.CourseNo LIKE 'IS%' AND OffYear = 2006
```

Debemos hacer notar que la extensión patentada de Oracle es inferior a la notación SQL:2003. La extensión patentada no permite la especificación del pedido mediante la realización de uniones externas. Esta limitación puede ser molesta en los problemas difíciles que comprenden más de una unión externa. Por tanto, debe usar el sintaxis de unión externa de SQL:2003, aunque las versiones más recientes de Oracle (9i y posteriores) todavía ofrecen soporte para la extensión patentada utilizando el símbolo (+).



# Capítulo 10

## Desarrollo de aplicaciones con vistas

### Objetivos de aprendizaje

Este capítulo describe los conceptos fundamentales para las vistas y demuestra su uso en formularios e informes. Al finalizar este capítulo, el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Escribir enunciados CREATE VIEW.
- Escribir consultas que utilicen vistas.
- Entender las ideas básicas acerca de los principios de modificación y materialización para procesar consultas con vistas.
- Aplicar reglas para determinar si es posible actualizar las vistas de una sola tabla o de tablas múltiples.
- Determinar los requisitos de datos para formularios jerárquicos.
- Escribir consultas que proporcionan datos para formularios jerárquicos.
- Formular consultas que proporcionen información para reportes jerárquicos.

### Panorama general

Los capítulos 3, 4 y 9 proporcionan el fundamento para entender las bases de datos relacionales y formular consultas en SQL. Lo más importante es que usted practicó con gran cantidad de ejemplos, adquirió habilidades para solucionar problemas de formulación de consultas y aprendió distintas partes de SQL. Este capítulo le muestra cómo aplicar sus habilidades para la formulación de consultas en la creación de aplicaciones con vistas.

Este capítulo enfatiza las vistas como la base para crear aplicaciones de bases de datos. Le proporcionamos los antecedentes esenciales para analizar el vínculo entre las vistas y las aplicaciones de bases de datos. Aprenderá la motivación para las vistas, el enunciado CREATE VIEW, el uso de vistas en SELECT y los enunciados de manipulación de datos (INSERT, UPDATE y DELETE). La mayor parte de los ejemplos de vistas en las secciones 10.2 y 10.3 tienen soporte en Microsoft Access como consultas almacenadas, y en Oracle como vistas. Después de estos antecedentes, aprenderá a utilizar las vistas para formularios y reportes jerárquicos. Va a aprender los pasos para analizar los requerimientos de datos que culminan en vistas para sopportarlos.

La presentación en las secciones 10.1 y 10.2 cubre las características de SQL:2003, que formaban parte de SQL-92. Algunas de las características de vistas actualizables en las secciones 10.3 y 10.4 son específicas para Microsoft Access debido al soporte variable con otros DBMS.

## 10.1 Antecedentes

---

### vista

una tabla derivada de una base o tablas físicas utilizando una consulta.

Una vista es una tabla virtual o derivada. Virtual significa que una vista se comporta como una tabla base pero físicamente no existe. Una vista se puede usar en una consulta como tabla base. Sin embargo, las filas de una vista no existen hasta que se derivan de las tablas base. Esta sección describe la razón de su importancia y cómo definirlas en SQL.

### 10.1.1 Motivación

Las vistas proporcionan el nivel externo de la arquitectura de tres esquemas descrita en el capítulo 1. Esta arquitectura promueve la independencia de datos para reducir el impacto de los cambios en la definición de bases de datos en las aplicaciones que las usan. Como es común hacer cambios en la definición de bases de datos, es importante reducir el impacto de estos cambios para controlar el costo de mantenimiento del software. Las vistas ofrecen soporte para la división en categorías de los requerimientos de las bases de datos, de modo que los cambios en su definición no afecten las aplicaciones que usan una vista. Si una aplicación tiene acceso a la base de datos a través de una vista, la mayoría de los cambios al esquema conceptual no van a afectar la aplicación. Por ejemplo, si cambia el nombre de tabla utilizado en una vista, también debe cambiar la definición de la vista, pero las aplicaciones que usan la vista no tienen que hacerlo.

Otro importante beneficio de las vistas es la simplificación de tareas. Se pueden formular muchas consultas con mayor facilidad si se usa una vista en lugar de las tablas base. Sin una vista, un enunciado SELECT puede comprender dos, tres o más tablas y requerir de la agrupación en caso de necesitar datos resumidos. Con una vista, el enunciado SELECT sólo puede referirse a una vista sin enlaces ni agrupaciones. Enseñar a los usuarios a escribir consultas con una sola tabla es mucho más sencillo que enseñarles a escribirlas con varias tablas y agrupaciones.

Las vistas ofrecen una simplificación similar a las macros en los lenguajes de programación y las hojas de cálculo. Una macro es un conjunto de comandos con un nombre. El hecho de usar una macro elimina la necesidad de especificar los comandos. De modo similar, el uso de una vista elimina el problema de escribir la consulta subyacente.

Las vistas ofrecen también un flexible nivel de seguridad. La restricción de acceso con vistas es más flexible que las restricciones para columnas y tablas, porque una vista es cualquier parte derivada de una base de datos. Los datos que no están en la vista permanecen ocultos para el usuario. Por ejemplo, es posible limitar a un usuario a departamentos, productos o regiones geográficas seleccionadas en una vista. La seguridad con el uso de tablas y columnas no puede especificar las condiciones y los cálculos, lo cual sí puede realizarse en una vista. Una vista puede incluir cálculos conjuntos para limitar a los usuarios a resúmenes de filas, en lugar de filas individuales.

La única desventaja de las vistas puede ser el desempeño. Para la mayoría de las vistas, su uso no comprende un compromiso considerable del desempeño en comparación con el uso directo de las tablas base. El uso de algunas vistas complejas puede comprender una reducción significativa del desempeño, en comparación con el uso directo de las tablas base. La baja en el desempeño puede variar dependiendo del DBMS. Antes de usar vistas complejas, le sugerimos comparar el desempeño con el uso de tablas.

### 10.1.2 Definición de vista

Definir una vista no es más difícil que escribir una consulta. SQL ofrece el enunciado CREATE VIEW, en el que se debe especificar el nombre de la vista, y el enunciado SELECT subyacente, tal como lo muestran los ejemplos 10.1 y 10.2. En Oracle se ejecuta directamente el enunciado CREATE VIEW.

En Microsoft Access, el enunciado CREATE VIEW se puede usar en el modo de consulta de SQL-92.<sup>1</sup> En el modo de consulta de SQL-89, la parte del enunciado SELECT en los ejemplos anteriores se puede guardar como una consulta almacenada para lograr el mismo efecto que en una vista. Es posible crear una consulta almacenada con sólo escribirla, darle un nombre y guardarla.

### EJEMPLO 10.1

#### Definición de una vista con tabla sencilla

Defina una vista llamada IS\_View que consista en estudiantes cuya materia principal sea IS.

```
CREATE VIEW IS_View AS
    SELECT * FROM Student
        WHERE StdMajor = 'IS'
```

StdSSN	StdFirstName	StdLastName	StdCity	StdState	StdZip	StdMajor	StdClass	StdGPA
123-45-6789	HOMER	WELLS	SEATTLE	WA	98121-1111	IS	FR	3.00
345-67-8901	WALLY	KENDALL	SEATTLE	WA	98123-1141	IS	SR	2.80
567-89-0123	MARIAH	DODGE	SEATTLE	WA	98114-0021	IS	JR	3.60
876-54-3210	CRISTOPHER	COLAN	SEATTLE	WA	98114-1332	IS	SR	4.00
890-12-3456	LUKE	BRAZZI	SEATTLE	WA	98116-0021	IS	SR	2.20
901-23-4567	WILLIAM	PILGRIM	BOTHEL	WA	98113-1885	IS	SO	3.80

### EJEMPLO 10.2

#### Definición de una vista con tabla múltiple

Defina una vista llamada MS\_View que consista en los cursos impartidos por los profesores del departamento de Ciencias Administrativas.

```
CREATE VIEW MS_View AS
    SELECT OfferNo, Offering.CourseNo, CrsUnits, OffTerm,
        OffYear, Offering.FacSSN, FacFirstName,
        FacLastName, OffTime, OffDays
    FROM Faculty, Course, Offering
        WHERE FacDept = 'MS'
            AND Faculty.FacSSN = Offering.FacSSN
            AND Offering.CourseNo = Course.CourseNo
```

OfferNo	CourseNo	CrsUnits	OffTerm	OffYear	FacSSN	FacFirstName	FacLastName	OffTime	OffDays
1234	IS320	4	FALL	2005	098-76-5432	LEONARD	VINCE	10:30 AM	MW
3333	IS320	4	SPRING	2006	098-76-5432	LEONARD	VINCE	8:30 AM	MW
4321	IS320	4	FALL	2005	098-76-5432	LEONARD	VINCE	3:30 PM	TTH
4444	IS320	4	WINTER	2006	543-21-0987	VICTORIA	EMMANUEL	3:30 PM	TTH
8888	IS320	4	SUMMER	2006	654-32-1098	LEONARD	FIBON	1:30 PM	MW
9876	IS460	4	SPRING	2006	654-32-1098	LEONARD	FIBON	1:30 PM	TTH
5679	IS480	4	SPRING	2006	876-54-3210	CRISTOPHER	COLAN	3:30 PM	TTH

Es probable que en el enunciado CREATE VIEW aparezca una lista de nombres de columna entre paréntesis después del nombre de la vista. Esta lista es necesaria cuando quiere renombrar dos o más columnas a partir de los nombres que se usan en la cláusula SELECT. La lista de columnas se omite en MS\_View porque no hay columnas cuyo nombre haya cambiado.

<sup>1</sup> SQL-89 es el modo de consulta predeterminado en Microsoft Access 2002 y 2003. Es posible cambiar el modo de consulta utilizando el separador Tables/Query en la ventana Options (Tools → Options ...).

La lista de columnas se necesita en el ejemplo 10.3 para renombrar la columna del cálculo global [COUNT(\*)]. En caso de renombrar una columna debe agregarse la lista completa con los nombres de las columnas.

### EJEMPLO 10.3

#### Definición de una vista con columnas renombradas

Defina una vista llamada Enrollment\_View que consista en los datos de los cursos y el número de estudiantes que participan en ellos.

```
CREATE VIEW Enrollment_View
(OfferNo, CourseNo, Term, Year, Instructor, NumStudents)
AS
SELECT Offering.OfferNo, CourseNo, OffTerm, OffYear,
       FacLastName, COUNT(*)
  FROM Offering, Faculty, Enrollment
 WHERE Offering.FacSSN = Faculty.FacSSN
   AND Offering.OfferNo = Enrollment.OfferNo
 GROUP BY Offering.OfferNo, CourseNo, OffTerm, OffYear,
          FacLastName
```

OfferNo	CourseNo	Term	Year	Instructor	NumStudents
1234	IS320	FALL	2005	VINCE	6
4321	IS320	FALL	2005	VINCE	6
5555	FIN300	WINTER	2006	MACON	2
5678	IS480	WINTER	2006	MILLS	5
5679	IS480	SPRING	2006	COLAN	6
6666	FIN450	WINTER	2006	MILLS	2
7777	FIN480	SPRING	2006	MACON	3
9876	IS460	SPRING	2006	FIBON	7

## 10.2 Uso de vistas para recuperación

Esta sección muestra ejemplos de consultas que usan vistas y explica el procesamiento de las mismas. Después de mostrar ejemplos en la sección 10.2.1, la sección 10.2.2 describe dos métodos para procesar consultas con vistas.

### 10.2.1 Uso de vistas en enunciados SELECT

Una vez que se ha definido la vista puede utilizarse en enunciados SELECT. Sólo tiene que usar el nombre de la vista en la cláusula FROM y las columnas en otras partes del enunciado. Es posible agregar otras condiciones y seleccionar un subconjunto de las columnas, como muestran los ejemplos 10.4 y 10.5.

### EJEMPLO 10.4 (Oracle)

#### Consulta con el uso de una vista de varias tablas

Elabore una lista con los cursos impartidos durante la primavera de 2006 en MS\_View.

```
SELECT OfferNo, CourseNo, FacFirstName, FacLastName,
       OffTime, OffDays
  FROM MS_View
 WHERE OffTerm = 'SPRING' AND OffYear = 2006
```

OfferNo	CourseNo	FacFirstName	FacLastName	OffTime	OffDays
3333	IS320	LEONARD	VINCE	8:30 AM	MW
9876	IS460	LEONARD	FIBON	1:30 PM	TTH
5679	IS480	CRISTOPHER	COLAN	3:30 PM	TTH

**EJEMPLO 10.5  
(Oracle)****Consulta con el uso de una vista agrupada**

Haga una lista con los cursos impartidos en la primavera de 2006 en IS en Enrollment\_View. En Access necesita sustituir el \* con % como símbolo comodín.

```
SELECT OfferNo, CourseNo, Instructor, NumStudents
  FROM Enrollment_View
 WHERE Term = 'SPRING' AND Year = 2006
       AND CourseNo LIKE 'IS%'
```

OfferNo	CourseNo	Instructor	NumStudents
5679	IS480	COLAN	6
9876	IS460	FIBON	7

Ambas consultas son mucho más fáciles de escribir que las originales. Quizás un usuario novato pueda escribir las dos con un poco de práctica. En contraste, este mismo usuario podría necesitar varias horas de aprendizaje para escribir consultas con varias tablas y agrupaciones.

De acuerdo con SQL:2003 es posible usar una vista en cualquier consulta. En la práctica, la mayor parte de los DBMS tienen algunas limitaciones en cuanto al uso de vistas en las consultas. Por ejemplo, algunos DBMS no ofrecen soporte para las consultas que muestran los ejemplos 10.6 y 10.7.<sup>2</sup>

**EJEMPLO 10.6  
(Oracle)****Consulta agrupada con el uso de una vista derivada de una consulta agrupada**

Elabore una lista con el número promedio de estudiantes por nombre de profesor utilizando Enrollment\_View.

```
SELECT Instructor, AVG(NumStudents) AS AvgStdCount
  FROM Enrollment_View
 GROUP BY Instructor
```

Instructor	AvgStdCount
COLAN	6
FIBON	7
MACON	2.5
MILLS	3.5
VINCE	6

**EJEMPLO 10.7  
(Oracle)****Enlace de una tabla base con una vista derivada de una consulta agrupada**

Haga una lista con el número de curso, el profesor, el número de estudiantes y las unidades del curso utilizando la vista Enrollment\_View y la tabla Course.

```
SELECT OfferNo, Instructor, NumStudents, CrsUnits
  FROM Enrollment_View, Course
```

<sup>2</sup>Microsoft Access 97 a 2003 y Oracle 8i a 10g ofrecen soporte para los ejemplos 10.6 y 10.7.

```
WHERE Enrollment_View.CourseNo = Course.CourseNo
AND NumStudents < 5
```

OfferNo	Instructor	NumStudents	CrsUnits
5555	MACON	2	4
6666	MILLS	2	4
7777	MACON	3	4

#### materialización de una vista

método para procesar una consulta en una vista mediante la ejecución directa de la consulta en la vista almacenada. Esta última se puede materializar con base en la demanda (al presentar la consulta de la vista), o reconstruirse en forma periódica a partir de las tablas base. Por lo general, en el caso de las bases de datos con una combinación de actividades de recuperación y actualización, la materialización no es una manera eficiente de procesar una consulta en una vista.

#### modificación de una vista

método para procesar una consulta en una vista que comprende la ejecución de una sola consulta. Una consulta que utiliza una vista se traduce en una consulta que usa tablas base al reemplazar las referencias a la vista con su definición. Para las bases de datos con una combinación de actividades de recuperación y actualización, la modificación ofrece la manera eficiente de procesar una consulta en una vista.

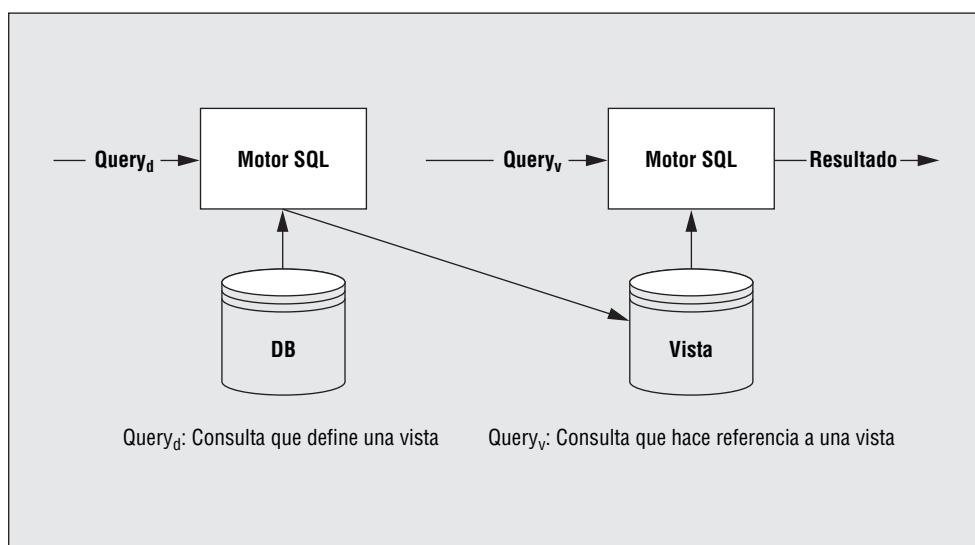
## 10.2.2 Procesamiento de consultas con referencias a vista

Para procesar consultas que hacen referencia a una vista, el DBMS puede emplear una estrategia de materialización o de modificación. La materialización de una vista requiere del almacenamiento de las filas de la vista. La forma más sencilla de almacenar una vista es crearla a partir de las tablas base de acuerdo con la demanda (al presentar la consulta de la vista). Para procesar una consulta con referencia a una vista es necesario que un DBMS ejecute dos consultas, como lo ilustra la figura 10.1. Un usuario presenta una consulta utilizando una vista (Query<sub>v</sub>). Se ejecuta la consulta que define la vista (Query<sub>d</sub>) y se crea una tabla de vista temporal. La figura 10.1 ilustra esta acción con la flecha en la vista. Después, se ejecuta la consulta que usa la vista empleando la tabla de vista temporal.

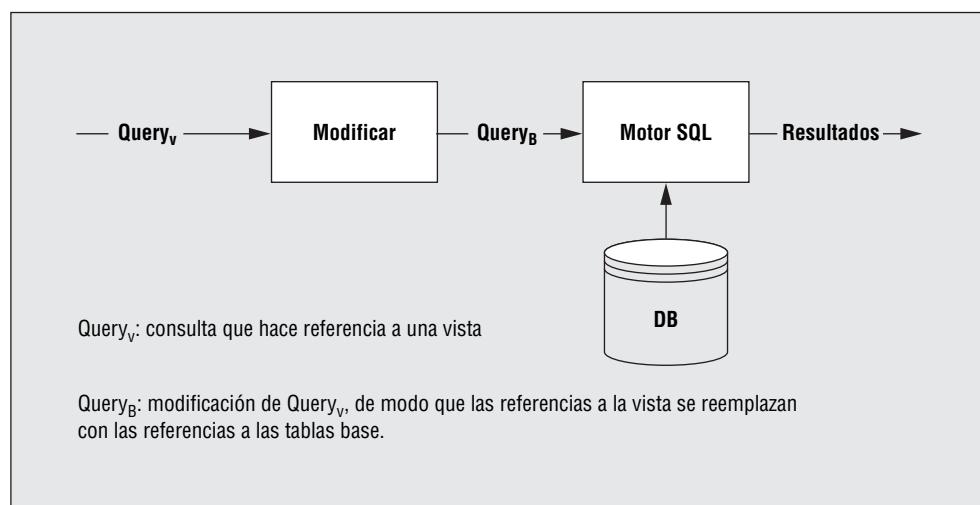
Por lo general, la materialización de la vista no es la estrategia preferida porque requiere que el DBMS ejecute dos consultas; sin embargo, en ciertas consultas es necesaria la materialización, como en los ejemplos 10.6 y 10.7. Además, es preferible la materialización en los *data warehouses* en los que dominan las recuperaciones. En un entorno *data warehouse*, las vistas se refrescan en forma periódica a partir de tablas de datos, en lugar de materializarse según la demanda. El capítulo 16 estudia las vistas materializadas que se utilizan en los *data warehouses*.

En un ambiente con una combinación de operaciones de actualización y recuperación, la modificación de una vista casi siempre ofrece un mejor desempeño que la materialización porque el DBMS sólo ejecuta una consulta. La figura 10.2 muestra que una consulta con el uso de una vista se modifica o vuelve a escribir como una consulta que sólo emplea tablas base; luego, la consulta modificada se ejecuta usando las tablas base. El proceso de modificación sucede en forma automática sin el conocimiento ni la acción de ningún usuario. En casi todos los DBMS es imposible ver la consulta modificada aun cuando quiera revisarla.

**FIGURA 10.1** Flujo del proceso de materialización de una vista



**FIGURA 10.2**  
Flujo del proceso de modificación de una vista



Como ejemplo de modificación de una vista, considere la transformación que se observa del ejemplo 10.8 al ejemplo 10.9. Al ejecutar una consulta utilizando una vista, la referencia a la vista se reemplaza con su definición. El nombre de la vista en la cláusula FROM cambia por las tablas base. Además, las condiciones en la cláusula WHERE se combinan utilizando el AND booleano con las condiciones de la consulta que definen la vista. Las partes subrayadas en el ejemplo 10.9 indican las sustituciones realizadas en el proceso de modificación.

#### EJEMPLO 10.8 Consulta con el uso de MS\_View

```
SELECT OfferNo, CourseNo, FacFirstName, FacLastName,
       OffTime, OffDays
  FROM MS_View
 WHERE OffTerm = 'SPRING' AND OffYear = 2006
```

OfferNo	CourseNo	FacFirstName	FacLastName	OffTime	OffDays
3333	IS320	LEONARD	VINCE	8:30 AM	MW
9876	IS460	LEONARD	FIBON	1:30 PM	TTH
5679	IS480	CRISTOPHER	COLAN	3:30 PM	TTH

#### EJEMPLO 10.9 Modificación del ejemplo 10.8

Se modifica el ejemplo 10.8 reemplazando las referencias a MS\_View con las referencias a la tabla base.

```
SELECT OfferNo, Course.CourseNo, FacFirstName,
       FacLastName, OffTime, OffDays
  FROM Faculty, Course, Offering
 WHERE FacDept = 'MS'
       AND Faculty.FacSSN = Offering.FacSSN
       AND Offering.CourseNo = Course.CourseNo
       AND OffTerm = 'SPRING' AND OffYear = 2006
```

Algunos DBMS llevan a cabo una simplificación adicional de las consultas modificadas para eliminar los enlaces innecesarios. Por ejemplo, la tabla *Course* no es necesaria porque en el ejemplo 10.9 no hay condiciones ni columnas de esa tabla. Además, el enlace entre las tablas *Offering* y *Course* no es necesario porque todas las filas de *Offering* se relacionan con una fila de *Course* (no se permite ninguna fila nula). Como resultado, es posible simplificar la consulta modificada eliminando la tabla *Course*. La simplificación da como resultado un tiempo de ejecución más breve, ya que el factor más importante para el tiempo de ejecución es el número de tablas.

### EJEMPLO 10.10 Simplificación mayor del ejemplo 10.9

Simplifique eliminando la tabla *Course* en el ejemplo 10.9 porque no es necesaria.

```
SELECT OfferNo, CourseNo, FacFirstName, FacLastName,
       OffTime, OffDays
  FROM Faculty, Offering
 WHERE FacDept = 'MS'
   AND Faculty.FacSSN = Offering.FacSSN
   AND OffTerm = 'SPRING' AND OffYear = 2006
```

## 10.3 Actualización con el uso de vistas

**vista de actualización**  
vista que se puede utilizar en enunciados SELECT, así como en enunciados UPDATE, INSERT y DELETE. Las vistas que sólo se pueden usar en enunciados SELECT se conocen como vistas de sólo lectura.

Dependiendo de su definición, una vista puede ser de sólo lectura o con capacidad de actualización. Una vista de sólo-lectura se puede usar en los enunciados SELECT, como se demuestra en la sección 10.2. Todas las vistas son por lo menos de sólo lectura. Una vista de este tipo no se puede usar en consultas que comprenden los enunciados SELECT, UPDATE y DELETE. Una vista que se puede usar tanto en los enunciados de modificación como en los enunciados SELECT se conoce como vista de actualización. Esta sección describe las reglas para definir las vistas de actualización con una sola y con varias tablas.

### 10.3.1 Vistas de actualización con tabla única

Una vista de actualización le permite insertar, actualizar o eliminar filas en las tablas base subyacentes realizando la operación correspondiente en la vista. Siempre que realice una modificación a una fila de una vista, la misma operación se lleva a cabo en la tabla base. De manera intuitiva esto significa que las filas en una vista de actualización corresponden a las filas de las tablas base subyacentes. Si una vista contiene la llave primaria de la tabla base, cada fila de la vista corresponde a una fila en la tabla base. Una vista de una tabla se puede actualizar si cumple con las siguientes tres reglas, que incluyen el requisito de la llave primaria.

#### Reglas para vistas de actualización con tabla única

1. La vista incluye la llave primaria de la tabla base.
2. En la vista están todos los campos requeridos (NOT NULL) de la tabla base sin un valor predeterminado.
3. La consulta de la vista no incluye las palabras clave GROUP BY ni DISTINCT.

Siguiendo estas reglas, *Fac\_View1* (ejemplo 10.11) se puede actualizar, mientras que *Fac\_View2* (ejemplo 10.12) y *Fac\_View3* (ejemplo 10.13) son de sólo lectura. *Fac\_View1* se puede actualizar suponiendo que las columnas *Faculty* que faltan no sean necesarias. *Fac\_View2* viola las reglas 1 y 2, mientras que *Fac\_View3* viola las tres reglas haciendo que ambas vistas sean de sólo lectura.

Como *Fac\_View1* se puede actualizar, es posible usarla con los enunciados INSERT, UPDATE y DELETE para cambiar la tabla *Faculty*. En el capítulo 4 utilizó estos enunciados para

**EJEMPLO 10.11 Vista de actualización con tabla única**

Cree una vista del subconjunto de filas y columnas con la llave primaria.

```
CREATE VIEW Fac_View1 AS
    SELECT FacSSN, FacFirstName, FacLastName, FacRank,
           FacSalary, FacDept, FacCity, FacState, FacZipCode
      FROM Faculty
     WHERE FacDept = 'MS'
```

FacSSN	FacFirstName	FacLastName	FacRank	FacSalary	FacDept	FacCity	FacState	FacZipCode
098-76-5432	LEONARD	VINCE	ASST	35000.00	MS	SEATTLE	WA	98111-9921
543-21-0987	VICTORIA	EMMANUEL	PROF	120000.00	MS	BOTHELL	WA	98011-2242
654-32-1098	LEONARD	FIBON	ASSC	70000.00	MS	SEATTLE	WA	98121-0094
876-54-3210	CRISTOPHER	COLAN	ASST	40000.00	MS	SEATTLE	WA	98114-1332

**EJEMPLO 10.12 Vista de sólo lectura con tabla única**

Cree un subconjunto de filas y columnas sin la llave primaria.

```
CREATE VIEW Fac_View2 AS
    SELECT FacDept, FacRank, FacSalary
      FROM Faculty
     WHERE FacSalary > 50000
```

FacDept	FacRank	FacSalary
MS	PROF	120000.00
MS	ASSC	70000.00
FIN	PROF	65000.00
FIN	ASSC	75000.00

**EJEMPLO 10.13 Vista de sólo-lectura con tabla única**

Cree una vista agrupada con el departamento y el salario promedio de los profesores.

```
CREATE View Fac_View3 (FacDept, AvgSalary) AS
    SELECT FacDept, AVG(FacSalary)
      FROM Faculty
     WHERE FacRank = 'PROF'
   GROUP BY FacDept
```

FacDept	AvgSalary
FIN	65000
MS	120000

cambiar las filas en las tablas base. Los ejemplos 10.14 a 10.16 demuestran que estos enunciados se pueden utilizar para cambiar las filas de la vista y de las tablas base subyacentes. Debemos notar que las modificaciones a las vistas están sujetas a las reglas de integridad de la tabla base subyacente. Por ejemplo, la inserción se rechaza en el ejemplo 10.14 si otra fila de *Faculty* tiene 999-99-8888 como número de seguridad social. Al eliminar las filas en una vista o cambiar la columna de la llave primaria, se aplican las reglas en las filas de referencia (sección 3.4).

Por ejemplo, se rechaza la eliminación en el ejemplo 10.16 si la fila *Faculty* con *FacSSN* 098-76-5432 tiene filas relacionadas con la tabla *Offering* y la regla de eliminación para la relación *Faculty-Offering* se establece en RESTRICT (restringir).

#### **EJEMPLO 10.14 Operación de inserción en una vista de actualización**

Inserte una nueva fila de profesores en el departamento MS.

```
INSERT INTO Fac_View1
(FacSSN, FacFirstName, FacLastName, FacRank, FacSalary,
FacDept, FacCity, FacState, FacZipCode)
VALUES ('999-99-8888', 'JOE', 'SMITH', 'PROF', 80000,
'MS', 'SEATTLE', 'WA', '98011-011')
```

#### **EJEMPLO 10.15 Operación de actualización en una vista de actualización**

Dé a los profesores asistentes en *Fac\_View1* un aumento de 10 por ciento.

```
UPDATE Fac_View1
SET FacSalary = FacSalary * 1.1
WHERE FacRank = 'ASST'
```

#### **EJEMPLO 10.16 Operación de eliminación en una vista de actualización**

Eliminar un profesor específico de *Fac\_View1*.

```
DELETE FROM Fac_View1
WHERE FacSSN = '999-99-8888'
```

#### *Actualizaciones de vista con efectos secundarios*

Algunas de las modificaciones a las vistas con capacidad de actualización pueden ser problemáticas, como lo muestran el ejemplo 10.17 y las tablas 10.1 y 10.2. El enunciado de actualización en el ejemplo 10.17 cambia el departamento de la última fila (Victoria Emmanuel) de la vista y la fila correspondiente en la tabla base. Sin embargo, al regenerar la vista, desaparece la fila que cambió (tabla 10.2). La actualización tiene el efecto secundario de hacer que la fila desaparezca de la vista. Este tipo de efecto puede ocurrir siempre que una columna con la cláusula WHERE de la definición de vista cambie por un enunciado UPDATE. El ejemplo 10.17 actualiza la columna *FactDept*, misma que se usa en la cláusula WHERE de la definición de la vista *Fac\_View1*.

#### **EJEMPLO 10.17 Operación de actualización en una vista de actualización provocando un efecto secundario**

Cambie el departamento de los profesores con salarios altos al departamento de finanzas.

```
UPDATE Fac_View1
SET FacDept = 'FIN'
WHERE FacSalary > 100000
```

**TABLA 10.1** Fac\_View1 antes de la actualización

FacSSN	FacFirstName	FacLastName	FacRank	FacSalary	FacDept	FacCity	FacState	FacZipCode
098-76-5432	LEONARD	VINCE	ASST	35000.00	MS	SEATTLE	WA	98111-9921
543-21-0987	VICTORIA	EMMANUEL	PROF	120000.00	MS	BOTHELL	WA	98011-2242
654-32-1098	LEONARD	FIBON	ASSC	70000.00	MS	SEATTLE	WA	98121-0094
876-54-3210	CRISTOPHER	COLAN	ASST	40000.00	MS	SEATTLE	WA	98114-1332

**TABLA 10.2** Fac\_View1 después de la actualización del ejemplo 10.17

FacSSN	FacFirstName	FacLastName	FacRank	FacSalary	FacDept	FacCity	FacState	FacZipCode
098-76-5432	LEONARD	VINCE	ASST	35000.00	MS	SEATTLE	WA	98111-9921
654-32-1098	LEONARD	FIBON	ASSC	70000.00	MS	SEATTLE	WA	98121-0094
876-54-3210	CRISTOPHER	COLAN	ASST	40000.00	MS	SEATTLE	WA	98114-1332

### EJEMPLO 10.18 Vista de actualización y tabla única con el uso de WITH CHECK OPTION (con opción de revisión)

Cree una vista de subconjunto con filas y columnas utilizando la llave primaria. La opción WITH CHECK OPTION no tiene soporte en Access.

```
CREATE VIEW Fac_View1_Revised AS
    SELECT FacSSN, FacFirstName, FacLastName, FacRank,
           FacSalary, FacDept, FacCity, FacState, FacZipCode
      FROM Faculty
     WHERE FacDept = 'MS'
  WITH CHECK OPTION
```

#### WITH CHECK OPTION

cláusula en el enunciado CREATE VIEW que evita los efectos secundarios al actualizar una vista. La cláusula WITH CHECK OPTION evita enunciados UPDATE e INSERT que violen la cláusula WHERE de una vista.

Como este efecto secundario puede ser confuso para un usuario, la cláusula WITH CHECK OPTION se puede usar para evitar las actualizaciones con efectos secundarios. Si WITH CHECK OPTION se especifica en el enunciado CREATE VIEW (ejemplo 10.18), se rechazan los enunciados INSERT y UPDATE que violan la cláusula WHERE. La actualización en el ejemplo 10.17 se rechaza si *Fac\_View1* contiene una cláusula CHECK OPTION, porque el cambio en *FacDept* a FIN viola la condición WHERE.

#### 10.3.2 Vistas de actualización y tabla múltiple

Quizá se sorprenda, pero algunas vistas con varias tablas también se pueden actualizar. Si la vista contiene la llave primaria de cada tabla, una vista con varias tablas puede corresponder en forma precisa con cada una de las filas de más de una tabla. Como las vistas con varias tablas son más complejas que aquellas que sólo tienen una, no existe un acuerdo general acerca de las reglas para su actualización.

Algunos DBMS no ofrecen soporte para actualizar vistas con varias tablas. Otros sistemas ofrecen soporte para actualizar una mayor cantidad de vistas con tablas. En esta sección describimos las reglas de actualización en Microsoft Access, ya que soporta gran cantidad de vistas con varias tablas. Además, las reglas para las vistas con capacidad de actualización en Access están relacionadas con la presentación de formas jerárquicas en la sección 10.4.

Para completar la presentación de las reglas de actualización de Access, el apéndice 10.B describe las reglas para las vistas con enlace que se pueden actualizar en Oracle. Las reglas para estas vistas en Oracle son similares a las de Microsoft Access, aunque Oracle es más restrictivo en las operaciones de manipulación permitidas y el número de tablas que es posible actualizar.

En Access, las consultas de tabla múltiple con soporte para actualizaciones se conocen como consultas 1-M que se pueden actualizar. Una consulta de este tipo comprende dos o más tablas; una de ellas representa el papel de tabla madre o tabla 1 y la otra representa el papel de hija o tabla M. Por ejemplo, en una consulta que comprende las tablas *Course* y *Offering*, *Course* representa el papel de madre y *Offering* el de hija. Para poder actualizar una consulta 1-M, siga estas reglas:

#### **Reglas para actualización de consultas 1-M**

1. La consulta incluye la llave primaria para la tabla hija.
2. La consulta contiene todas las columnas requeridas (NOT NULL) sin valores predeterminados para la tabla hija.
3. La consulta no incluye GROUP BY ni DISTINCT.
4. La columna de enlace en la tabla madre debe ser única (ya sea una llave primaria o una limitación única).
5. La consulta contiene la(s) columna(s) de la llave foránea de la tabla hija.
6. Si la vista soporta las operaciones de inserción en la tabla madre, la consulta incluye la llave primaria y las columnas requeridas por la tabla madre. Las operaciones de actualización tienen soporte en la tabla madre, aun cuando la consulta no contenga la llave primaria de la tabla madre.

Con el uso de estas reglas, es posible actualizar *Course\_Offering\_View1* (ejemplo 10.19) y *Faculty\_Offering\_View1* (ejemplo 10.2). No se puede actualizar *Course\_Offering\_View2* (ejemplo 10.20) porque falta *Offering.CourseNo* (la llave foránea en la tabla hija). En los enunciados SELECT se emplea el estilo del operador de unión (palabras clave INNER JOIN) porque Microsoft Access lo pide para las peticiones 1-M que se puedan actualizar.

#### **EJEMPLO 10.19 Consulta 1-M que se puede actualizar (Access)**

Cree una consulta 1-M con capacidad de actualización (guardada como *Course\_Offering\_View1*) con un enlace entre las tablas *Course* y *Offering*.

##### **Course\_Offering\_View1:**

```
SELECT Course.CourseNo, CrsDesc, CrsUnits,
       Offering.OfferNo, OffTerm, OffYear,
       Offering.CourseNo, OffLocation, OffTime, FacSSN,
       OffDays
  FROM Course INNER JOIN Offering
    ON Course.CourseNo = Offering.CourseNo
```

#### **EJEMPLO 10.20 Consulta de sólo-lectura con tablas múltiples (Access)**

Esta consulta (guardada como *Course\_Offering-View2*) es de sólo-lectura porque no contiene *Offering.CourseNo*.

##### **Course\_Offering\_View2:**

```
SELECT CrsDesc, CrsUnits, Offering.OfferNo,
       Course.CourseNo, OffTerm, OffYear, OffLocation,
       OffTime, FacSSN, OffDays
  FROM Course INNER JOIN Offering
    ON Course.CourseNo = Offering.CourseNo
```

**EJEMPLO 10.21 Consulta 1-M que se puede actualizar  
(Access)**

Cree una consulta 1-M de actualización (guardada como Faculty\_Offering\_View1) con un enlace entre las tablas *Faculty* y *Offering*.

**Faculty\_Offering\_View1:**

```
SELECT Offering.OfferNo, Offering.FacSSN, CourseNo,
       OffTerm, OffYear, OffLocation, OffTime, OffDays,
       FacFirstName, FacLastName, FacDept
  FROM Faculty INNER JOIN Offering
    ON Faculty.FacSSN = Offering.FacSSN
```

*Inserción de filas en consultas 1-M que se pueden actualizar*

La inserción de una fila nueva en una consulta 1-M que se puede actualizar es más compleja que insertar una fila en una vista tabla única. Esta complicación se presenta porque existe una opción relacionada con las tablas que soportan operaciones de inserción. Como resultado de una actualización de la vista es posible insertar solamente las filas de la tabla hija o de las tablas hija y madre. Para insertar una fila en la tabla hija sólo deben proporcionar los valores necesarios para esta operación, como se muestra en el ejemplo 10.22. Observe que el valor para *Offering.CourseNo* y *Offering.FacSSN* debe coincidir con las filas existentes en las tablas *Course* y *Faculty*, respectivamente.

**EJEMPLO 10.22 Inserción de una fila en la tabla hija como resultado de una actualización de vista  
(Access)**

Inserción de una fila nueva en *Offering* como resultado de usar *Course\_Offering\_View1*.

```
INSERT INTO Course_Offering_View1
( Offering.OfferNo, Offering.CourseNo, OffTerm, OffYear,
  OffLocation, OffTime, FacSSN, OffDays )
VALUES ( 7799, 'IS480', 'SPRING', 2000, 'BLM201',
         '#1:30PM#, '098-76-5432', 'MW' )
```

Para insertar una fila en ambas tablas (madre e hija), la vista debe incluir la llave primaria y las columnas necesarias de la tabla madre. El hecho de proporcionar los valores para todas las columnas inserta una fila en ambas tablas siempre y cuando la vista incluya estas columnas, tal como se muestra en el ejemplo 10.23. Proporcionar los valores sólo para la tabla madre inserta una fila sólo en esta tabla, como muestra el ejemplo 10.24. En ambos ejemplos, el valor para *Course.CourseNo* no debe coincidir con una fila existente en *Course*.

**EJEMPLO 10.23 Inserción de una fila en ambas tablas como resultado de una actualización de vista  
(Access)**

Inserción de una fila nueva en *Course* y *Offering* como resultado del uso de *Course\_Offering\_View1*.

```
INSERT INTO Course_Offering_View1
( Course.CourseNo, CrsUnits, CrsDesc, Offering.OfferNo,
  OffTerm, OffYear, OffLocation, OffTime, FacSSN,
  OffDays )
VALUES ( 'IS423', 4, 'OBJECT ORIENTED COMPUTING', 8877,
         'SPRING', 2006, 'BLM201', '#3:30PM#,
         '123-45-6789', 'MW' )
```

**EJEMPLO 10.24 Inserción de una fila en la tabla madre como resultado de una actualización de vista**

Inserción de una fila nueva en la tabla *Course* como resultado del uso de *Course\_Offering\_View1*.

```
INSERT INTO Course_Offering_View1
( Course.CourseNo, CrsUnits, CrsDesc)
VALUES ( 'IS481', 4, 'ADVANCED DATABASE' )
```

*Consultas 1-M que se pueden actualizar con más de dos tablas*

También se pueden actualizar las consultas que comprenden más de dos tablas. Aplicamos las mismas reglas a las consultas 1-M con más de dos tablas. Sin embargo, es preciso aplicar las reglas a cada enlace en la consulta. Por ejemplo, si una consulta tiene tres tablas (dos enlaces), las reglas se aplican a ambos enlaces. En *Faculty\_Offering\_Course\_View1* (ejemplo 10.25), *Offering* es la tabla hija en ambos enlaces. Por tanto, las llaves foráneas (*Offering.CourseNo* y *Offering.FacSSN*) deben estar en el resultado de la consulta. En *Faculty\_Offering\_Course\_Enrollment\_View1* (ejemplo 10.26), *Enrollment* es la tabla hija en un enlace y *Offering* es la tabla hija en los otros dos enlaces. La llave primaria de la tabla *Offering* no es necesaria en el resultado, a menos que las filas de *Offering* se tengan que insertar utilizando la vista. La consulta en el ejemplo 10.26 ofrece soporte para inserciones en la tabla *Enrollment* y actualizaciones en las otras tablas.

Las reglas específicas sobre las operaciones de inserción, actualización y eliminación que tienen soporte en las consultas 1-M que se pueden actualizar son más complejas de lo que aquí

**EJEMPLO 10.25 Consulta 1-M con tres tablas que se puede actualizar  
(Access)**

**Faculty\_Offering\_Course\_View1:**

```
SELECT CrsDesc, CrsUnits, Offering.OfferNo,
       Offering.CourseNo, OffTerm, OffYear, OffLocation,
       OffTime, Offering.FacSSN, OffDays, FacFirstName,
       FacLastName
  FROM ( Course INNER JOIN Offering
         ON Course.CourseNo = Offering.CourseNo )
    INNER JOIN Faculty
         ON Offering.FacSSN = Faculty.FacSSN
```

**EJEMPLO 10.26 Consulta 1-M con cuatro tablas que se puede actualizar  
(Access)**

**Faculty\_Offering\_Course\_Enrollment\_View1:**

```
SELECT CrsDesc, CrsUnits, Offering.CourseNo,
       Offering.FacSSN, FacFirstName, FacLastName,
       OffTerm, OffYear, OffLocation, OffTime, OffDays,
       Enrollment.OfferNo, Enrollment.StdSSN,
       Enrollment.EnrGrade
  FROM ( ( Course INNER JOIN Offering
         ON Course.CourseNo = Offering.CourseNo )
    INNER JOIN Faculty
         ON Offering.FacSSN = Faculty.FacSSN )
    INNER JOIN Enrollment
         ON Enrollment.OfferNo = Offering.OfferNo
```

describimos. El propósito de esta sección es demostrar la complejidad de las vistas con varias tablas que se pueden actualizar y sus reglas. La documentación de Microsoft Access ofrece una descripción completa de las reglas.

Las opciones relacionadas con las tablas que se pueden actualizar en una consulta 1-M pueden ser confusas, sobre todo cuando la consulta incluye más de dos tablas. Por lo general, sólo se puede actualizar la tabla hija, de modo que no aplican las consideraciones en los ejemplos 10.23 y 10.24. Las opciones casi siempre se basan en las necesidades de los formularios de captura de datos, que estudiaremos en la siguiente sección.

## 10.4 Uso de vistas en formularios jerárquicos

Uno de los beneficios más importantes de las vistas es que constituyen los bloques de construcción para las aplicaciones. Los formularios de captura de datos, piedra angular de casi todas las aplicaciones de base de datos, ofrecen soporte para la recuperación y modificación de tablas. Los formularios de captura de datos tienen formatos que los hacen visualmente atractivos y fáciles de usar. En contraste, el formato estándar de los resultados de las consultas no es atractivo para muchos usuarios. Esta sección describe el formulario jerárquico, un tipo importante de formulario de captura de datos, así como las relaciones entre las vistas y los formularios jerárquicos.

### 10.4.1 ¿Qué es un formulario jerárquico?

**formulario jerárquico**  
ventana con formato para la captura y presentación de datos utilizando una parte fija (formulario principal) y otra variable (subformulario). El formulario principal muestra un registro y el subformulario presenta varios registros relacionados.

Un formulario es un documento que se utiliza en un proceso de negocios. Está diseñado para soportar una tarea de negocios, como el procesamiento de un pedido, la inscripción en una clase o la reservación en una línea aérea. Los formularios jerárquicos ofrecen soporte para tareas de negocios con una parte fija y otra variable. La parte fija del formulario jerárquico se conoce como formulario principal, mientras que la parte variable (que se repite) se llama subformulario. Por ejemplo, un formulario jerárquico para los cursos ofrecidos (figura 10.3) muestra la información de los cursos en el formulario principal y los datos sobre los cursos ofrecidos en el subformulario. Un formulario jerárquico para el registro de las clases (figura 10.4) muestra los datos de registro y de los estudiantes en el formulario principal y las inscripciones en los cursos ofrecidos en el subformulario. Los campos para el cálculo de facturas debajo del subformulario son parte del formulario principal. En cada formulario, el subformulario puede mostrar varios registros, mientras que el principal sólo muestra uno.

**FIGURA 10.3**  
Ejemplo de formulario de cursos ofrecidos

**FIGURA 10.4**  
Ejemplo de formulario de registro

Los formularios jerárquicos pueden ser parte de un sistema de formularios relacionados. Por ejemplo, un sistema de información de estudiantes puede tener formularios para admisiones, registro de calificaciones, aprobación de cursos, horario de cursos y asignación de profesores a los cursos. Estos formularios pueden estar relacionados en forma indirecta a través de actualizaciones a la base de datos o de manera directa mediante el envío de datos entre ellos. Por ejemplo, las actualizaciones a la base de datos que se realizan procesando un formulario de registro se utilizan al final de un semestre en un formulario de registro de calificaciones. Este capítulo enfatiza los requisitos de datos para cada formulario, una habilidad importante para el desarrollo de aplicaciones. Esta habilidad complementa otras capacidades para el desarrollo de aplicaciones, como el diseño de interfaces de usuario y de flujos de trabajo.

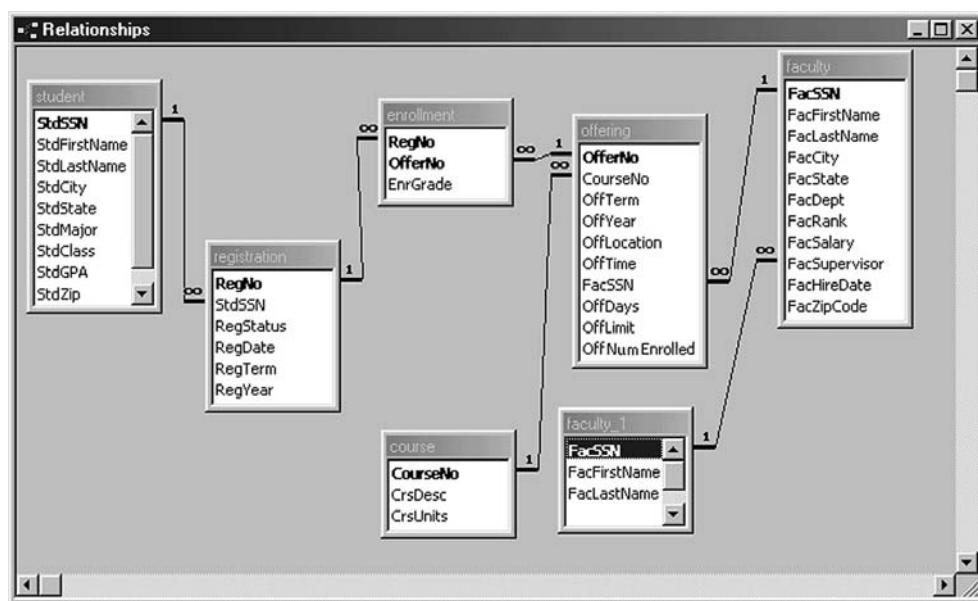
#### 10.4.2 Relación entre formularios jerárquicos y tablas

Los formularios jerárquicos ofrecen soporte para operaciones en las relaciones 1-M. Una jerarquía o árbol es una estructura con relaciones 1-M. Cada relación 1-M tiene una tabla madre (tabla 1) y una tabla hija (tabla M). Un formulario jerárquico le permite al usuario insertar, actualizar, eliminar y recuperar registros en ambas tablas de la relación 1-M. Un formulario jerárquico está diseñado para manipular (desplegar, insertar, actualizar y eliminar) la tabla madre en el formulario principal y la tabla hija en el subformulario. En esencia, un formulario jerárquico es una interfaz conveniente para las operaciones en una relación 1-M.

A manera de ejemplo, considere los formularios jerárquicos que se muestran en las figuras 10.3 y 10.4. En Course Offering Form (figura 10.3), la relación entre las tablas *Course* y *Offering* permite que el formulario despliegue una fila *Course* en el formulario principal, y filas *Offering* relacionadas en el subformulario. Registration Form (figura 10.4) opera en las tablas *Registration* y *Enrollment*, así como la relación 1-M entre éstas. La tabla *Registration* es una nueva tabla en la base de datos de la universidad. La figura 10.5 muestra un diagrama revisado de esta relación.

Para respaldar un proceso de negocios a menudo es útil mostrar otra información en el formulario principal y el subformulario. Por lo general, esta información (externa a las tablas madre e hija) es para fines de visualización. Aunque es posible diseñar un formulario para permitir cambios en las columnas de otras tablas, los requerimientos particulares del proceso de negocios quizás no garantizan esta posibilidad. Por ejemplo, Registration Form (figura 10.4) contiene columnas de la tabla *Student*, de modo que es posible autenticar al usuario. De modo similar, las columnas de las tablas *Offering*, *Faculty* y *Course* se muestran en el subformulario,

**FIGURA 10.5**  
Relaciones en la base de datos revisada de la universidad



de modo que el usuario puede tomar decisiones de inscripción con base en la información. Cuando un proceso de negocios permite cambios en las columnas de otras tablas se debe utilizar otro formulario para completar esta tarea.

#### 10.4.3 Habilidades de formulación de consultas para formularios jerárquicos

Para implementar un formulario jerárquico es preciso tomar decisiones para cada uno de los pasos enlistados a continuación. Estos pasos ayudan a aclarar la relación entre un formulario y las tablas de la base de datos relacionada. Además, se pueden utilizar directamente para implementar el formulario en algunos DBMS, como Microsoft Access.

1. Identificar la relación 1-M que manipula el formulario.
2. Identificar las columnas de enlace o unión para la relación 1-M.
3. Identificar otras tablas en el formulario principal y el subformulario.
4. Determinar la capacidad de actualización de las tablas en el formulario jerárquico.
5. Escribir las consultas para el formulario principal y el subformulario.

##### Paso 1: Identificar la relación 1-M

La decisión más importante es comparar el formulario con una relación 1-M en la base de datos. Si empieza por una imagen del formulario (como la figura 10.3 o 10.4), busque una relación que contenga columnas de la tabla madre en el formulario principal y columnas de la tabla hija en el subformulario. Por lo regular, la tabla madre contiene la llave primaria del formulario principal. En la figura 10.3, el campo Course No. es la llave primaria del formulario principal, de modo que la tabla Course es la tabla madre. En la figura 10.4, el campo Registration No. es la llave primaria del formulario principal, de modo que la tabla Registration es la tabla madre. Si usted mismo va a realizar el diseño y la distribución del formulario, tome una decisión en cuanto a la relación 1-M antes de trazar su distribución.

##### Paso 2: Identificar las columnas de unión

Si puede identificar la relación 1-M, casi siempre le será fácil identificar las columnas de unión. Estas últimas son simplemente las columnas de enlace (*join*) de ambas tablas (madre e hija) en la relación. En la figura 10.3, las columnas de unión son *Course.CourseNo* y *Offering.CourseNo*. En la figura 10.4, las columnas de unión son *Registration.RegNo* y *Enrollment.RegNo*. Es

importante recordar que las columnas de unión conectan el formulario principal con el subformulario. Con esta conexión el subformulario sólo muestra las filas que coinciden con el valor de la columna de unión del formulario principal. Sin esta conexión, el subformulario despliega todas las filas y no sólo las relacionadas con el registro desplegado en el formulario principal.

#### *Paso 3: Determinar otras tablas*

Además de la relación 1-M, es posible mostrar otras tablas en el formulario principal y el subformulario para proporcionarle un contexto al usuario. Si observa columnas de otras tablas, deberá tomar nota de ellas con el fin de poder usarlas en el paso 5, cuando escriba las consultas para el formulario. Por ejemplo, Registration Form incluye columnas de la tabla *Student* en el formulario principal. El subformulario incluye columnas de las tablas *Offering*, *Faculty* y *Course*. No nos vamos a ocupar de las columnas calculadas, como Total Units, hasta implementar el formulario.

#### *Paso 4: Determinar las tablas que se pueden actualizar*

Para el cuarto paso es necesario identificar las tablas que puede cambiar cuando utilice el formulario. Por lo general, sólo hay una tabla en el formulario principal y otra en el subformulario que necesita cambiar cuando el usuario capture los datos. En Registration Form, la tabla *Registration* cambia cuando el usuario manipula el formulario principal, y *Enrollment* cambia cuando manipula el subformulario. Las tablas identificadas en el paso 3 generalmente son de sólo-lectura. En el subformulario Registration las tablas *Student*, *Offering*, *Faculty* y *Course* son de sólo-lectura. Para algunos campos de formulario que no se actualizan en el formulario jerárquico, es posible usar botones para transferir otro formulario con el fin de poder cambiar los datos. Por ejemplo, es posible agregar un botón al formulario principal para permitirle al usuario cambiar los datos de los estudiantes en otro formulario.

Hay ocasiones en que el formulario principal no ofrece soporte para la actualización en ninguna tabla. En Course Offering Form, la tabla *Course* no cambia al utilizar el formulario principal. La razón de que el formulario principal sea de sólo-lectura es para soportar el proceso de aprobación de cursos. Casi todas las universidades requieren de un proceso de aprobación por separado para los nuevos cursos usando un formulario independiente. Course Offering Form está diseñado sólo para agregar ofertas a los cursos existentes. Si una universidad no tiene esta limitación, se puede usar el formulario principal para cambiar la tabla *Course*.

Como parte del diseño de un formulario jerárquico, es preciso entender con claridad los requisitos subyacentes del proceso de negocios. Después, estos requisitos se deben transformar en decisiones acerca de las tablas que se ven afectadas por las acciones del usuario en el formulario, como la actualización de un campo o la inserción de un nuevo registro.

#### *Paso 5: Escribir consultas de formulario*

El último paso integra las decisiones tomadas en los pasos anteriores. Es preciso escribir una consulta para el formulario principal y otra para el subformulario. Estas consultas deben soportar las actualizaciones de las tablas que identificó en el paso 4. Es necesario seguir las reglas para formular vistas de actualización (con una y varias tablas) que dimos en la sección 10.3. Algunos DBMS pueden solicitarle el uso de un enunciado CREATE VIEW para estas consultas, mientras que otros le permiten escribir directamente los enunciados SELECT.

Las tablas 10.3 y 10.4 resumen las respuestas a los cuatro primeros pasos para los formularios Course Offering y Registration. Para el paso 5, los ejemplos 10.27 a 10.30 muestran las consultas para los formularios principales y los subformularios de las figuras 5.3 y 5.4. En el

**TABLA 10.3**  
**Resumen de los pasos para la formulación de consultas para el formulario Course Offering**

Paso	Respuesta
1	<i>Course</i> (tabla madre), <i>Offering</i> (tabla hija)
2	<i>Course.CourseNo</i> , <i>Offering.CourseNo</i>
3	Sólo datos de las tablas <i>Course</i> y <i>Offering</i>
4	Operaciones insertar, actualizar y eliminar en la tabla <i>Offering</i> del subformulario

**TABLA 10.4**

**Resumen de los pasos para la formulación de consultas para el formulario Registration**

Paso	Respuesta
1	<i>Registration</i> (tabla madre), <i>Enrollment</i> (tabla hija)
2	<i>Registration.RegNo</i> , <i>Enrollment.RegNo</i>
3	Datos de la tabla <i>Student</i> en el formulario principal y las tablas <i>Offering</i> , <i>Course</i> y <i>Faculty</i> en el subformulario
4	Operaciones insertar, actualizar y eliminar en la tabla <i>Registration</i> del formulario principal, y en la tabla <i>Enrollment</i> del subformulario

en el ejemplo 10.29, el campo *Address* del formulario (figura 10.4) se deriva de las columnas *StdCity* y *StdState*. En el ejemplo 10.30 no se necesita la llave primaria de la tabla *Offering* porque la consulta no ofrece soporte para insertar operaciones en la tabla *Offering*. La consulta sólo soporta la inserción de operaciones en la tabla *Enrollment*. Observe que todos los ejemplos siguen las reglas de Microsoft Access (versiones 97 a 2003) para consultas 1-M que se puedan actualizar.

**EJEMPLO 10.27 Consulta para el formulario principal del formulario Course Offering (Access)**

```
SELECT CourseNo, CrsDesc, CrsUnits FROM Course
```

**EJEMPLO 10.28 Consulta para el subformulario del formulario Course Offering (Access)**

```
SELECT * FROM Offering
```

**EJEMPLO 10.29 Consulta para el formulario principal del formulario Registration (Access)**

```
SELECT RegNo, RegTerm, RegYear, RegDate,
       Registration.StdSSN, RegStatus, StdFirstName,
       StdLastName, StdClass, StdCity, StdState
  FROM Registration INNER JOIN Student
    ON Registration.StdSSN = Student.StdSSN
```

**EJEMPLO 10.30 Consulta para el subformulario del formulario Registration (Access)**

```
SELECT RegNo, Enrollment.OfferNo, Offer.CourseNo, OffTime,
       OffLocation, OffTerm, OffYear, Offering.FacSSN,
       FacFirstName, FacLastName, CrsDesc, CrsUnits
  FROM ( ( Enrollment INNER JOIN Offering
            ON Enrollment.OfferNo = Offering.OfferNo )
        INNER JOIN Faculty
            ON Faculty.FacSSN = Offering.FacSSN )
        INNER JOIN Course
            ON Course.CourseNo = Offering.CourseNo
```

Existe un problema adicional en la consulta del subformulario Registration Form (ejemplo 10.30). La consulta del subformulario despliega una fila *Offering* sólo si hay una fila *Faculty* relacionada. Si quiere que el subformulario despliegue las filas *Offering* sin importar si hay una fila *Faculty* relacionada o no, es necesario usar un enlace exterior de un lado (*one-sided outer join*), como muestra el ejemplo 10.31. Usted puede saber si es necesario un enlace exterior con

sólo ver copias del formulario. Si encuentra cursos ofrecidos sin profesor asignado, necesita un enlace exterior de un lado en la consulta.

**EJEMPLO 10.31 Consulta revisada del subformulario con un enlace exterior de un lado**
**(Access)**

```
SELECT RegNo, Enrollment.OfferNo, Offering.CourseNo,
       OffTime, OffLocation, OffTerm, OffYear,
       Offering.FacSSN, FacFirstName, FacLastName,
       CrsDesc, CrsUnits
  FROM ( ( Enrollment INNER JOIN Offering
            ON Enrollment.OfferNo = Offering.OfferNo )
    INNER JOIN Course
            ON Offering.CourseNo = Course.CourseNo )
    LEFT JOIN Faculty
            ON Faculty.FacSSN = Offering.FacSSN
```

A manera de ejemplo, la tabla 10.5 resume las respuestas a los pasos de la formulación de consultas para Faculty Assignment Form que muestra la figura 10.6. El objetivo de este formulario es ofrecer soporte a los administradores para asignar profesores a los cursos ofrecidos. La relación 1-M en el formulario es la relación de la tabla *Faculty* con la tabla *Offering*. Este formulario no se puede usar para insertar las nuevas filas *Faculty* ni cambiar datos sobre *Faculty*. Además, tampoco se puede utilizar para insertar nuevas filas *Offering*. La única operación de actualización que tiene soporte en este formulario es cambiar los profesores asignados para impartir un curso existente. Los ejemplos 10.32 y 10.33 muestran las consultas del formulario principal y el subformulario.

**TABLA 10.5**  
Resumen de los pasos para la formulación de consultas para el formulario Faculty Assignment

Paso	Respuesta
1	<i>Faculty</i> (tabla madre), <i>Offering</i> (tabla hija)
2	<i>Faculty.FacSSN</i> , <i>Offering.FacSSN</i>
3	Sólo datos de las tablas <i>Faculty</i> y <i>Offering</i>
4	Actualizar <i>Offering.FacSSN</i>

**FIGURA 10.6**  
Ejemplo del formulario Faculty Assignment

**EJEMPLO 10.32 Consulta del formulario principal para el formulario Faculty Assignment (Access)**

```
SELECT FacSSN, FacFirstName, FacLastName, FacDept
FROM Faculty
```

**EJEMPLO 10.33 Consulta del subformulario para el formulario Faculty Assignment (Access)**

```
SELECT OfferNo, Offering.CourseNo, FacSSN, OffTime,
OffDays, OffLocation, CrsUnits
FROM Offering INNER JOIN COURSE
ON Offering.CourseNo = Course.CourseNo
```

## 10.5 Uso de vistas en los reportes

Además de ser los bloques para la construcción de los formularios de captura de datos, las vistas son también el fundamento de los reportes. Un reporte es una presentación estilizada de los datos apropiados para una audiencia seleccionada. Un reporte es similar a un formulario en el sentido de que ambos utilizan vistas y presentan los datos de modo muy diferente a como aparecen en las tablas base. Un reporte difiere de un formulario en que el primero no cambia las tablas base, mientras que el formulario sí puede realizar cambios en ellas. Esta sección describe el reporte jerárquico, un tipo de reporte muy poderoso, y su relación con las vistas.

**reporte jerárquico**

despliegue con formato de una consulta usando indentado para mostrar la información agrupada y clasificada.

### 10.5.1 ¿Qué es un reporte jerárquico?

Los reportes jerárquicos (también conocidos como reportes con corte de control) utilizan el anidamiento o indentado o la tabulación para ofrecer un formato visualmente atractivo. El reporte de agenda de profesores (figura 10.7) muestra los datos ordenados por departamento, nombre

**FIGURA 10.7** Reporte de agenda de profesores

Reporte de agenda de profesores para el año académico 2005-2006							
Department	Name	Term	Course No.	Offer No.	Days	Start Time	Location
FIN	MACON, NICKI	SPRING					
			FIN480	7777	MW	1:30 PM	BLM305
		WINTER					
	MILLS, JULIA		FIN300	5555	MW	8:30 AM	BLM207
			Líneas de detalles				
		WINTER					
			FIN450	6666	TTH	10:30 AM	BLM212
			IS480	5678	MW	10:30 AM	BLM302
MS	COLAN, CRISTOPHER	SPRING					

del profesor y trimestre. Cada campo tabulado se conoce como grupo. El anidamiento de los grupos indica el orden de clasificación del reporte. La línea más interiorizada en un reporte se conoce como línea de detalle. En el reporte de agenda de profesores, las líneas de detalle muestran el número de curso, el número de ofrecimiento y otros detalles del curso asignado. Estas líneas también se pueden clasificar. En el reporte de agenda de profesores, las líneas de detalle se encuentran clasificadas por número de curso.

La principal ventaja de los reportes jerárquicos es que los usuarios entienden mejor el significado de los datos cuando son clasificados y ordenados en forma tabulada. Es difícil inspeccionar el resultado estándar de una consulta (una hoja de cálculo) cuando el resultado contiene datos de varias tablas. Por ejemplo, compare el reporte de agenda de profesores con la hoja de cálculo que muestra la misma información (figura 10.8). Puede crear confusión la repetición del departamento, nombre del profesor y equipo.

Para mejorar la apariencia, los reportes jerárquicos pueden mostrar datos resumidos en las líneas de detalle, columnas computadas y cálculos después de los grupos. Las líneas de detalle en la figura 10.9 muestran las inscripciones (número de estudiantes inscritos) en cada curso impartido por un profesor. En SQL el número de estudiantes se calcula con la función COUNT. Se calculan las columnas Percent Full [(Enrollment/Limit) \* 100%] y Low Enrollment (valor verdadero/falso). Un cuadro de verificación es una manera visualmente atractiva para desplegar las columnas verdadero/falso. Muchos reportes muestran cálculos resumidos al final de cada grupo. En el reporte de carga de trabajo para los profesores, los cálculos resumidos muestran el total de unidades y estudiantes, así como el porcentaje promedio de cursos completos ofrecidos.

**FIGURA 10.8** Hoja de cálculo que muestra el contenido del reporte de agenda de profesores

FacDept	FacLastName	FacFirstName	OffTerm	CourseNo	OfferNo	OffLocation	OffTime	OffDays
FIN	MACON	NICKI	SPRING	FIN480	7777	BLM305	1:30 PM	MW
FIN	MACON	NICKI	WINTER	FIN300	5555	BLM207	8:30 AM	MW
FIN	MILLS	JULIA	WINTER	FIN450	6666	BLM212	10:30 AM	TTH
FIN	MILLS	JULIA	WINTER	IS480	5678	BLM302	10:30 AM	MW
MS	COLAN	CRISTOPHER	SPRING	IS480	5679	BLM412	3:30 PM	TTH
MS	EMMANUEL	VICTORIA	WINTER	IS320	4444	BLM302	3:30 PM	TTH
MS	FIBON	LEONARD	SPRING	IS460	9876	BLM307	1:30 PM	TTH
MS	VINCE	LEONARD	FALL	IS320	4321	BLM214	3:30 PM	TTH
MS	VINCE	LEONARD	FALL	IS320	1234	BLM302	10:30 AM	MW
MS	VINCE	LEONARD	SPRING	IS320	3333	BLM214	8:30 AM	MW

**FIGURA 10.9** Reporte de carga de trabajo para los profesores

Faculty Work Load Report for the 2005–2006 Academic Year								
Department	Name	Term	Offer Number	Units	Limit	Enrollment	Percent Full	Low Enrollment
FIN	JULIA MILLS	WINTER	5678	4	20	1	5.00%	<input checked="" type="checkbox"/>
		Líneas de detalles	Summary for 'term' = WINTER (1 detail record)					
			Sum	4	1			
			Avg			5.00%		
			Summary for JULIA MILLS					
			Sum	4	1			
			Avg			5.00%		
			Summary for 'department' = FIN (1 detail record)					

**consejo para la formulación de consultas en reportes jerárquicos**  
 la consulta para un reporte debe producir datos para las líneas de detalle. Si las líneas de detalle de un reporte contienen datos resumidos, la consulta deberá contener también datos resumidos.

### 10.5.2 Habilidades para la formulación de consultas en reportes jerárquicos

La formulación de consultas para los reportes es similar a la utilizada para los formularios. Al formular una consulta para un reporte debe (1) comparar los campos en el reporte con las columnas de la base de datos, (2) determinar las tablas necesarias y (3) identificar las condiciones de enlace. La mayoría de las consultas para reportes comprenden enlaces, y quizás hasta enlaces externos de un lado. No son comunes las consultas complejas que comprenden operaciones de diferencia y división. Estos pasos pueden seguirse para formular la consulta mostrada en el ejemplo 10.34, para el reporte de agenda de profesores (figura 10.7).

Formular consultas para reportes jerárquicos es, en cierta forma, más fácil que para formularios jerárquicos. Las consultas para los reportes no necesitan tener capacidad de actualización (casi siempre son de sólo-lectura). Además, sólo hay una consulta para un reporte, mientras que hay dos o más consultas para un formulario jerárquico.

#### EJEMPLO 10.34

##### Consulta para el reporte de agenda de profesores

```
SELECT Faculty.FacSSN, Faculty.FacFirstName, FacLastName,
       Faculty.FacDept, Offering.OfferNo,
       Offering.CourseNo, Offering.OffTerm,
       Offering.OffYear, Offering.OffLocation,
       Offering.OffTime, Offering.OffDays
  FROM Faculty, Offering
 WHERE Faculty.FacSSN = Offering.FacSSN
   AND ( ( Offering.OffTerm = 'FALL'
          AND Offering.OffYear = 2005 )
        OR ( Offering.OffTerm = 'WINTER'
          AND Offering.OffYear = 2006 )
        OR ( Offering.OffTerm = 'SPRING'
          AND Offering.OffYear = 2006 ) )
```

El principal problema en la formulación de consultas para reportes jerárquicos es el nivel del resultado. En ocasiones hay una opción para el resultado de la consulta si contiene filas individuales o agrupadas. Por regla general la consulta debe producir datos para las líneas de detalle del reporte. La consulta para el reporte de carga de trabajo para los profesores (ejemplo 10.35) agrupa los datos y cuenta el número de estudiantes inscritos. La consulta produce directamente datos para las líneas de detalle del reporte. Si la consulta produce una fila por estudiante inscrito

#### EJEMPLO 10.35

##### Consulta para el reporte de carga de trabajo para los profesores con datos resumidos en las líneas de detalle

```
SELECT Offering.OfferNo, FacFirstName, FacLastName,
       FacDept, OffTerm, CrsUnits, OffLimit,
       Count(Enrollment.RegNo) AS NumStds,
       NumStds/Offlimit AS PercentFull,
       (NumStds/Offlimit) < 0.25 AS LowEnrollment
  FROM Faculty, Offering, Course, Enrollment
 WHERE Faculty.FacSSN = Offering.FacSSN
   AND Course.CourseNo = Offering.CourseNo
   AND Offering.OfferNo = Enrollment.OfferNo
   AND ( ( Offering.OffTerm = 'FALL'
          AND Offering.OffYear = 2005 ) )
```

```

    OR ( Offering.OffTerm = 'WINTER'
    AND Offering.OffYear = 2006 )
    OR ( Offering.OffTerm = 'SPRING'
    AND Offering.OffYear = 2006 )
GROUP BY Offering.OfferNo, FacFirstName, FacLastName,
        FacDept, OffTerm, CrsUnits, OffLimit

```

en un curso (un mayor nivel de detalle), el reporte debe calcular el número de estudiantes inscritos. Con la mayoría de las herramientas de reporte es más fácil realizar cálculos conjuntos en la consulta cuando la línea de detalle muestra sólo datos resumidos.

Los otros cálculos del ejemplo 10.35 (*PercentFull* y *LowEnrollment*) se pueden llevar a cabo en la consulta o el reporte prácticamente con el mismo esfuerzo. Observe que el campo *OffLimit* es una columna nueva en la tabla *Offering* y muestra el número máximo de estudiantes que se pueden inscribir en un curso ofrecido.

## Reflexión final

Este capítulo describe las vistas, que son tablas virtuales derivadas de tablas base con consultas. Los conceptos importantes acerca de las vistas son la motivación para éstas y su uso en el desarrollo de aplicaciones de bases de datos. El principal beneficio de las vistas es la independencia de los datos. Los cambios a las definiciones de la tabla base por lo general no afectan a las aplicaciones que utilizan vistas. Éstas también pueden simplificar las consultas escritas por los usuarios, al tiempo que ofrecen una unidad flexible para el control de la seguridad. Para utilizar las vistas de manera efectiva necesita entender la diferencia entre las vistas de sólo-lectura y aquéllas con capacidad de actualización. Una vista de sólo-lectura se puede usar en una consulta de la misma manera que una tabla base. Todas las vistas son por lo menos de sólo-lectura, pero algunas se pueden actualizar. Con una vista que es posible actualizar, los cambios en la fila de la vista se propagan a las tablas base subyacentes. Es posible actualizar las vistas con una y con varias tablas. El factor más importante para determinar la capacidad de actualización es que una vista contenga llaves primarias de las tablas base subyacentes.

Las vistas son los bloques de construcción de las aplicaciones de bases de datos porque son utilizadas por los formularios y reportes. Los formularios de captura de datos ofrecen soporte para la recuperación y cambios a la bases de datos. Los formularios jerárquicos manipulan las relaciones 1-M en una base de datos. Para definir un formulario jerárquico, necesita identificar la relación 1-M y definir vistas que se puedan actualizar para la parte fija (formulario principal) y variable (subformulario) del formulario. Los reportes jerárquicos ofrecen una presentación de datos visualmente atractiva. Para definir un reporte de este tipo debe identificar los niveles de agrupación y formular una consulta con el fin de obtener datos para las líneas detalladas del reporte.

Este capítulo continúa la parte 5 enfatizando el desarrollo de aplicaciones con bases de datos de relación. En el capítulo 9 amplió sus habilidades para la formulación de consultas y la comprensión de las bases de datos de relación empezó en los capítulos de la parte 2. Este capítulo enfatiza la aplicación de las habilidades para la formulación de consultas en la creación de aplicaciones basadas en vistas. El capítulo 11 demuestra el uso de consultas en procedimientos almacenados para personalizar y extender las aplicaciones de bases de datos. Para reafirmar sus conocimientos sobre el desarrollo de aplicaciones con vistas necesita utilizar un DBMS relacional, especialmente para crear formularios y reportes. Sólo va a comprender totalmente los conceptos al utilizarlos en una aplicación real de bases de datos.

## Revisión de conceptos

- Beneficios de las vistas: independencia de datos, formulación de consultas simplificadas, seguridad.
- Definición de vista en SQL:

```

CREATE VIEW IS_Students AS
SELECT * FROM Student WHERE StdMajor = 'IS'

```

- Uso de una vista en una consulta:

```
SELECT StdFirstName, StdLastName, StdCity, StdGPA
  FROM IS_Students
 WHERE StdGPA >= 3.7
```

- Uso de una vista que se puede actualizar en los enunciados INSERT, UPDATE y DELETE:

```
UPDATE IS_Students
  SET StdGPA = 3.5
 WHERE StdClass = 'SR'
```

- Modificación de una vista: servicio del DBMS para procesar una consulta en una vista que comprende la ejecución de una sola consulta. Una consulta que utiliza una vista se traduce en una consulta que usa tablas base reemplazando las referencias a la vista con su definición.
- Materialización de una vista: servicio del DBMS para procesar una consulta en una vista al ejecutarla directamente en la vista almacenada. Esta última se puede materializar según la demanda (al presentar la consulta de la vista) o reconstruirse en forma periódica a partir de las tablas base.
- Uso típico de la modificación de vistas para bases de datos con combinación de operaciones de actualización y recuperación.
- Vista que se puede actualizar: vista que es posible usar en enunciados SELECT, UPDATE, INSERT y DELETE.
- Reglas para definir vistas que se pueden actualizar con una tabla: llave primaria y columnas necesarias.
- Opción WITH CHECK para evitar actualizaciones de vistas con efectos secundarios.
- Reglas para definir vistas que se pueden actualizar con varias tablas: llave primaria y columnas necesarias de cada tabla que se pueda actualizar, además de llaves foráneas para las tablas hijas.
- Consultas 1-M que se pueden actualizar para desarrollar formulario de captura de datos en Microsoft Access.
- Componentes de un formulario jerárquico: formulario principal y subformulario.
- Formularios jerárquicos que ofrecen una interfaz conveniente para manipular relaciones 1-M.
- Pasos para la formulación de consultas en formularios jerárquicos: identificar la relación 1-M, localizar las columnas de enlace, ubicar otras tablas en el formulario, determinar la capacidad de actualización de las tablas, escribir las consultas del formulario.
- Escribir consultas que se pueden actualizar para el formulario principal y el subformulario.
- Reporte jerárquico: despliegue de una consulta con formato mediante el uso del indentado o tabulación para mostrarla agrupada y clasificada.
- Componentes de reportes jerárquicos: campos de agrupación, líneas de detalle y cálculos del resumen de grupos.
- Escribir consultas para reportes jerárquicos: proporcionar datos para las líneas de detalle.

## Preguntas

1. ¿De qué forma las vistas ofrecen una independencia de datos?
2. ¿De qué manera las vistas simplifican las consultas que escriben los usuarios?
3. ¿En qué se parece una vista a una macro en una hoja de cálculo?
4. ¿Qué es la materialización de una vista?
5. ¿Qué es la modificación de una vista?
6. ¿Cuándo es preferible la modificación a la materialización para procesar consultas de vista?
7. ¿Qué es una vista que se puede actualizar?

8. ¿Por qué algunas vistas son de sólo-lectura?
9. ¿Cuáles son las reglas para las vistas con una tabla que se pueden actualizar?
10. ¿Cuáles son las reglas para las consultas 1-M que se pueden actualizar en Microsoft Access? ¿Y para las vistas con varias tablas que se pueden actualizar?
11. ¿Cuál es el propósito de la cláusula WITH CHECK?
12. ¿Qué es un formulario jerárquico?
13. Describa en forma breve cómo se puede usar un formulario jerárquico en un proceso de negocios que usted conozca. Por ejemplo, si conoce algo sobre procesamiento de pedidos, describa de qué manera un formulario jerárquico podría apoyar este proceso.
14. ¿Cuál es la diferencia entre un formulario principal y un subformulario?
15. ¿Cuál es el propósito de las columnas de enlace en los formularios jerárquicos?
16. ¿Por qué debe escribir consultas que se puedan actualizar para un formulario principal y un subformulario?
17. ¿Por qué se usan las tablas en un formulario jerárquico aun cuando no se puedan cambiar como resultado del uso del formulario?
18. ¿Cuál es el primer paso del proceso de formulación de consultas para formularios jerárquicos?
19. ¿Cuál es el segundo paso del proceso de formulación de consultas para formularios jerárquicos?
20. ¿Cuál es el tercer paso del proceso de formulación de consultas para formularios jerárquicos?
21. ¿Cuál es el cuarto paso del proceso de formulación de consultas para formularios jerárquicos?
22. ¿Cuál es el quinto paso del proceso de formulación de consultas para formularios jerárquicos?
23. Mencione un ejemplo de un formulario jerárquico en el que el formulario principal no se pueda actualizar. Explique la razón de negocios que determina la condición de sólo lectura del formulario principal.
24. ¿Qué es un reporte jerárquico?
25. ¿Qué es una columna de agrupación en un reporte jerárquico?
26. ¿Cómo identifica las columnas de agrupación en un reporte?
27. ¿Qué es una línea de detalle en un reporte jerárquico?
28. ¿Cuál es la relación de la agrupación de columnas con las columnas de clasificación en un reporte?
29. ¿Por qué a menudo es más fácil escribir una consulta para un reporte jerárquico que para un formulario jerárquico?
30. ¿Qué significa que una consulta deba producir datos para la línea de detalle de un reporte jerárquico?
31. ¿Los DBMS comerciales están de acuerdo con las reglas para las vistas con varias tablas que se pueden actualizar? Si no es así, comente en forma breve el nivel de acuerdo acerca de las reglas para las vistas con varias tablas que se pueden actualizar.
32. ¿Qué efectos secundarios ocurren cuando un usuario cambia la fila de una vista que se puede actualizar? ¿Cuál es la causa de estos efectos secundarios?

## Problemas

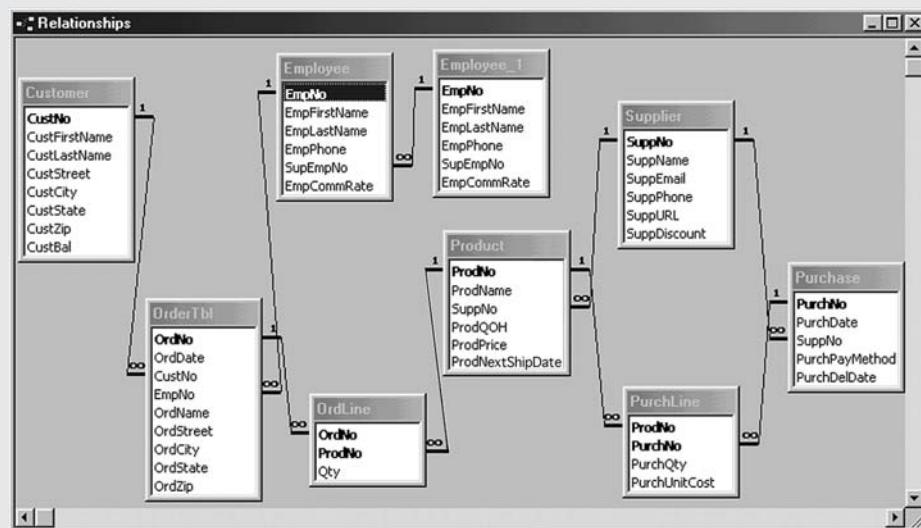
Los problemas utilizan la base de datos extendida de captura de pedidos que ilustran la figura 10.P1 y la tabla 10.P1. Los enunciados CREATE TABLE de Oracle para las tablas nuevas y la tabla *Product* revisada continúan la tabla 10.P1.

Esta base de datos amplía con tres tablas la base de datos de captura de pedidos utilizada en los problemas de los capítulos 3 y 9: (1) *Supplier*, que contiene la lista de proveedores de los productos manejados en el inventario; (2) *Purchase*, que registra los detalles generales de las compras para reabastecer el inventario, y (3) *PurchLine*, que contiene los productos pedidos en una compra. Además, la base de datos extendida de captura de pedidos contiene una nueva relación 1-M (*Supplier* con *Product*) que reemplaza la columna *Product.ProdMfg* en la base de datos original.

Además de las revisiones señaladas en el párrafo anterior, debe tener en cuenta varias suposiciones hechas en el diseño de la base de datos extendida de captura de pedidos:

- El diseño supone que sólo hay un proveedor para cada producto. Esta suposición es apropiada para una sola tienda detallista que hace el pedido directamente a los fabricantes.
- La relación 1-M de *Supplier* con *Purchase* apoya el proceso de compra. En este proceso, un usuario designa un proveedor antes de seleccionar los artículos que le va a pedir. Sin esta relación, sería más difícil implementar el proceso de negocios y los formularios de captura de datos relacionados.

**FIGURA 10.P1**  
Diagrama de relación para la base de datos revisada de entradas de pedidos



**TABLA 10.P1**  
Explicaciones de las columnas seleccionadas en la base de datos revisada de entradas de pedidos

Nombre de columna	Descripción
PurchDate	Fecha en que se hizo la compra
PurchPayMethod	Método de pago para la compra (crédito, PO o efectivo)
PurchDelDate	Fecha de entrega esperada para la compra
SuppDiscount	Descuento ofrecido por el proveedor
PurchQty	Cantidad de producto comprado
PurchUnitCost	Costo unitario del producto comprado

```
CREATE TABLE Product
(
    ProdNo          CHAR(8),
    ProdName        VARCHAR2(50) CONSTRAINT ProdNameRequired NOT NULL,
    SuppNo          CHAR(8) CONSTRAINT SuppNo1Required NOT NULL,
    ProdQOH         INTEGER DEFAULT 0,
    ProdPrice       DECIMAL(12,2) DEFAULT 0,
    ProdNextShipDate DATE,
    CONSTRAINT PKProduct PRIMARY KEY (ProdNo),
    CONSTRAINT SuppNoFK1 FOREIGN KEY (SuppNo) REFERENCES Supplier
    ON DELETE CASCADE
)
```

```
CREATE TABLE Supplier
(
    SuppNo          CHAR(8),
    SuppName        VARCHAR2(30) CONSTRAINT SuppNameRequired NOT NULL,
    SuppEmail       VARCHAR2(50),
    SuppPhone       CHAR(13),
    SuppURL         VARCHAR2(100),
    SuppDiscount    DECIMAL(3,3),
    CONSTRAINT PKSupplier PRIMARY KEY (SuppNo)
)
```

```
CREATE TABLE Purchase
(
    PurchNo         CHAR(8),
    PurchDate       DATE CONSTRAINT PurchDateRequired NOT NULL,
    SuppNo          CHAR(8) CONSTRAINT SuppNo2Required NOT NULL,
    PurchPayMethod  CHAR(6) DEFAULT 'PO',
    PurchDelDate    DATE,
    CONSTRAINT PKPurchase PRIMARY KEY (PurchNo),
    CONSTRAINT SuppNoFK2 FOREIGN KEY (SuppNo) REFERENCES Supplier
)
```

```

CREATE TABLE PurchLine
(
    ProdNo      CHAR(8),
    PurchNo     CHAR(8),
    PurchQty    INTEGER DEFAULT 1 CONSTRAINT PurchQtyRequired NOT NULL,
    PurchUnitCost DECIMAL(12,2),
CONSTRAINT PKPurchLine PRIMARY KEY (PurchNo, ProdNo),
CONSTRAINT FKPurchase FOREIGN KEY (PurchNo) REFERENCES Purchase
    ON DELETE CASCADE,
CONSTRAINT FKProdNo2 FOREIGN KEY (ProdNo) REFERENCES Product )

```

1. Defina una vista que contenga los productos del proveedor número S3399214. Incluya todas las columnas *Product*.
2. Defina una vista que contenga los detalles de los pedidos hechos en enero de 2007. Incluya todas las columnas *OrderTbl*, *OrdLine.Qty* y el número de producto.
3. Defina una vista que contenga el número de producto, nombre, precio y cantidad disponible de producto, además del número de pedidos en los que aparece dicho producto.
4. Utilizando la vista definida en el problema 1, escriba una consulta que mencione los productos con un precio mayor que 300 dólares. Incluya en el resultado todas las columnas de la vista.
5. Utilizando la vista definida en el problema 2, escriba una consulta que mencione las filas que contienen las palabras Ink Jet en el nombre de producto. Incluya en el resultado todas las columnas de la vista.
6. Utilizando la vista definida en el problema 3, escriba una consulta que mencione los productos para los que se han hecho más de cinco pedidos. Incluya en el resultado el nombre de producto y el número de pedidos realizados.
7. Modifique la consulta en el problema 4 de manera que sólo utilice tablas base.
8. Modifique la consulta en el problema 5 de manera que sólo utilice tablas base.
9. Modifique la consulta en el problema 6 de manera que sólo utilice tablas base.
10. ¿Se puede actualizar la vista del problema 1? Explique por qué sí o por qué no.
11. ¿Se puede actualizar la vista del problema 2? Explique por qué sí o por qué no. ¿Qué tablas de la base de datos pueden cambiarse modificando las filas en la vista?
12. ¿Se puede actualizar la vista en el problema 3? Explique por qué sí o por qué no.
13. Para la vista del problema 1, escriba un enunciado INSERT que haga referencia a la vista. El efecto del enunciado INSERT debe agregar una fila nueva a la tabla *Product*.
14. Para la vista del problema 1, escriba un enunciado UPDATE que haga referencia a la vista. El efecto del enunciado UPDATE debe modificar la columna *ProdQOH* de la fila agregada en el problema 13.
15. Para la vista del problema 1, escriba un enunciado DELETE que haga referencia a la vista. El efecto del enunciado DELETE deberá eliminar la fila que se agregó en el problema 13.
16. Modifique la definición de la vista del problema 1 para evitar efectos secundarios. Para esta vista utilice un nombre distinto al que usó en el problema 1. Observe que la opción WITH CHECK no puede especificarse en Microsoft Access utilizando la ventana SQL.
17. Escriba un enunciado UPDATE para la vista del problema 1 con el fin de modificar *SuppNo* en la fila con *ProdNo* de P6677900 a S4420948. La definición de la vista revisada en el problema 16 deberá rechazar el enunciado UPDATE, pero la definición de la vista original en el problema 1 deberá aceptarlo. Este problema no se puede resolver en Access utilizando la ventana SQL.
18. Defina una consulta 1-M que se pueda actualizar que comprenda las tablas *Customer* y *OrderTbl*. La consulta debe soportar actualizaciones a la tabla *OrderTbl*; además de incluir todas las columnas de la tabla *OrderTbl* y nombre, calle, ciudad, estado y código postal de la tabla *Customer*. Hay que hacer notar que este problema es específicamente para Microsoft Access.
19. Defina una consulta 1-M que se pueda actualizar y comprenda las tablas *Customer*, *OrderTbl* y *Employee*. La consulta debe soportar actualizaciones a la tabla *OrderTbl*, además de incluir todas las filas de la tabla *OrderTbl*, aun cuando haya un número de empleado nulo. Además, debe incluir todas las columnas de la tabla *OrderTbl*, nombre, calle, ciudad, estado y código postal de la tabla *Customer*, así como nombre y teléfono de la tabla *Employee*. Hay que hacer notar que este problema es específicamente para Microsoft Access.

20. Defina una consulta 1-M que se pueda actualizar y comprenda las tablas *OrdLine* y *Product*. La consulta debe soportar actualizaciones de la tabla *OrdLine* e incluir todas las columnas de esta tabla, así como nombre, cantidad disponible y precio de la tabla *Product*. Hay que señalar que este problema es específicamente para Microsoft Access.
21. Defina una consulta 1-M que se pueda actualizar que comprenda las tablas *Purchase* y *Supplier*. La consulta debe soportar actualizaciones e inserciones a las tablas *Product* y *Supplier*, e incluir todas las columnas necesarias para poder actualizar ambas tablas. Hay que señalar que este problema es específicamente para Microsoft Access.
22. Para la muestra del formulario simple de pedidos que aparece en la figura 10.P2, responda las cinco preguntas sobre la formulación de consultas estudiadas en la sección 10.4.3. El formulario ofrece soporte para la manipulación del encabezado y de los detalles de pedidos.
23. Para la muestra del formulario de pedidos que se ilustra en la figura 10.P3, responda las cinco preguntas sobre la formulación de consultas estudiadas en la sección 10.4.3. Al igual que el formulario simple de pedidos que aparece en la figura 10.P2, este formulario ofrece soporte para la manipulación del encabezado y los detalles de los pedidos. Además, el formulario de pedido muestra datos de otras tablas para proporcionar un contexto para el usuario al llenar un pedido. El formulario de pedidos

**FIGURA 10.P2**  
Formulario simple de pedidos

OrdNo	0123123	OrdStreet	9825 S. Crest Lane
OrdDate	01/23/2007	OrdCity	Bellevue
CustNo	C9432910	OrdState	WA
EmpNo	E9954302	OrdZip	98104-2211
OrdName	Larry Styles		

Simple Order Subform

	ProdNo	Qty
▶	P0036566	1
	P1445671	1
*		1

Record: [navigation buttons] 1 of 20

**FIGURA 10.P3**  
Formulario de pedidos

OrdNo	01231231	OrdDate	01/23/2007	CustNo	C9432910
OrdName:	Larry Styles	CustFirstName	Larry		
OrdStreet	9825 S. Crest Lane	OrdCity	Bellevue	CustLastName	Styles
OrdState	WA	OrdZip	98104-2211	CustStreet	9825 S. Crest Lane
EmpNo	E9954302	CustCity	Bellevue		
EmpFirstName	Mary	CustState	WA		
EmpLastName	Hill	CustZip:	98104-2211		

Order Details

ProdNo	Product	SuppNo	Supplier	Qty	Price	Amount
▶ P0036566	17 inch Color Monitor	S2029929	ColorMeg, Inc.	1	\$169.00	\$169.00
P1445671	8-Outlet Surge Protector	S4298800	Intersafe	1	\$14.99	\$14.99
*						

Total Amount: \$183.99

Record: [navigation buttons] 1 of 20

ofrece soporte tanto para pedidos por teléfono (un empleado toma el pedido) como por la web (sin que un empleado tome el pedido). La consulta del subformulario debe calcular el campo Amount como  $Qty * ProdPrice$ . No calcule el campo Total Amount en la consulta del formulario principal ni en la consulta del subformulario, ya que se va a calcular en el formulario.

24. Modifique su respuesta al problema 23 suponiendo que el formulario de pedidos sólo ofrezca soporte para pedidos por teléfono y no por la web.
25. Para el ejemplo Formulario simple de compras que aparece en la figura 10.P4, responda las cinco preguntas sobre la formulación de consultas estudiadas en la sección 10.4.3. El formulario ofrece soporte para la manipulación del encabezado y los detalles de las compras.
26. Para el ejemplo formulario de compras que se ilustra en la figura 10.P5, utilice los cinco pasos de la formulación de consultas estudiadas en la sección 10.4.3. Al igual que el formulario simple de

**FIGURA 10.P4**  
Formulario simple de compras

**Simple Purchase Form**

PurchNo	P2224040
PurchDate	02/03/2007
SuppNo	S2029929
PurchPayMethod	Credit
PurchDelDate	02/08/2007

**Purchase Subform**

	ProdNo	PurchQty	PurchUnitCost
▶	P0036566	10	\$100.00
	P0036577	10	\$200.00
*		1	\$0.00

Record: [◀◀] [▶▶] [\*] of 5

**FIGURA 10.P5**  
Formulario de compras

**Purchase Form**

PurchNo	P2224040	SuppName	ColorMeg, Inc.
PurchDate	02/03/2007	SuppEmail	custrel@colormeg.com
SuppNo	S2029929	SuppPhone	(720) 444-1231
PurchPayMethod	Credit	SuppURL	www.colormeg.com
PurchDelDate	02/08/2007		

**Purchase Lines**

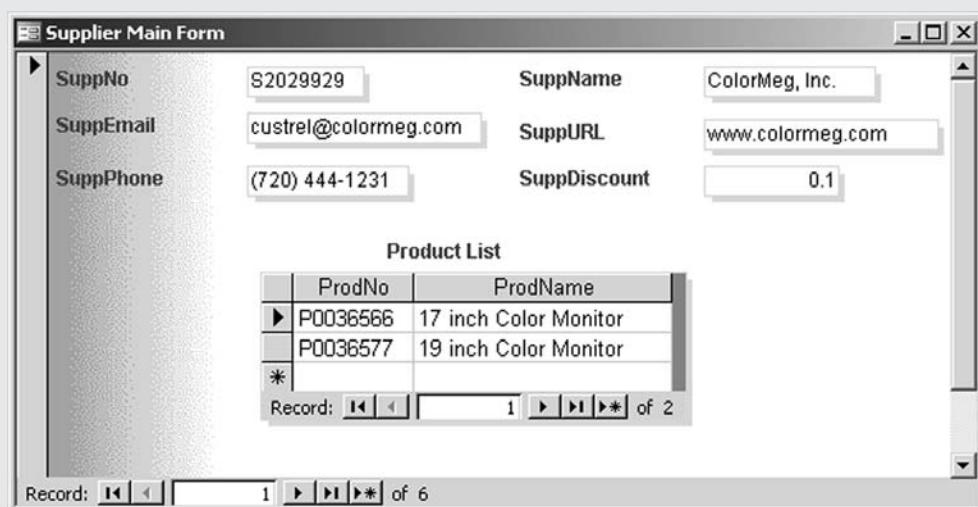
ProdNo	Product	QOH	Selling Price	Purchase Qty	Unit Cost	Amount
▶ P0036566	17 inch Color Monitor	12	\$169.00	10	\$100.00	\$1,000.00
P0036577	19 inch Color Monitor	10	\$319.00	10	\$200.00	\$2,000.00
*						

Record: [◀◀] [▶▶] [\*] of 2

Total Amount \$3,000.00

Record: [◀◀] [▶▶] [\*] of 5

**FIGURA 10.P6**  
Formulario de proveedores



compras que ilustra la figura 10.P4, el formulario de compras ofrece soporte para la manipulación del encabezado y los detalles de las compras. Además, el formulario de compras muestra datos de otras tablas para proporcionar un contexto para el usuario al completar la compra. La consulta del subformulario debe calcular el campo Amount como *PurchQty*\**PurchUnitCost*. No calcule el campo Total Amount en la consulta del formulario principal ni en la consulta del subformulario, ya que se va a calcular en el formulario.

27. Para el ejemplo del formulario de proveedores de la figura 10.P6, siga los cinco pasos para la formulación de consultas estudiadas en la sección 10.4.3. El formulario principal ofrece soporte para la manipulación de datos sobre los proveedores, mientras que el subformulario soporta sólo la manipulación del número y nombre de los productos que ofrece el proveedor en el formulario principal.
28. Para el reporte de detalles de pedidos, escriba un enunciado SELECT con el fin de generar los datos para las líneas de detalle. La columna de agrupación en el reporte es *OrdNo*. El reporte debe incluir los pedidos para el número de cliente O2233457 en enero de 2007.

#### Reporte de detalles de pedidos

Order Number	Order Date	Product No	Qty	Price	Amount
O2233457	1/12/2007	P1441567	1	\$14.99	\$14.99
		P0036577	2	\$319.00	\$638.00
Total Order Amount					\$652.99
O4714645	1/11/2007	P9995676	2	\$89.00	\$178.00
		P0036566	1	\$369.00	\$369.00
Total Order Amount					\$547.00

29. Para el ejemplo del reporte de resumen de pedidos, escriba un enunciado SELECT con el fin de producir datos para las líneas de detalle. El campo Zip Code del reporte son los primeros cinco caracteres de la columna *CustZip*. El campo de agrupamiento del reporte son los primeros cinco caracteres de la columna *CustZip*. El campo Order Amount Sum del reporte es el resultado de la cantidad por el precio del producto. Establezca el límite el reporte para los pedidos hechos en el año 2007. También deberá incluir el número de mes en el enunciado SELECT, de modo que el reporte se pueda clasificar por el número de mes, en lugar del campo Month. Utilice las expresiones siguientes para derivar las columnas calculadas que se usaron en el reporte:
  - En Microsoft Access, la expresión *left(CustZip, 5)* genera el campo Zip Code en el reporte. En Oracle, la expresión *substr(CustZip, 1, 5)* genera el campo Zip Code en el reporte.
  - En Microsoft Access, la expresión *format(OrdDate, "mmmm yyyy")* genera el campo Month en el reporte. En Oracle, la expresión *to\_char(OrdDate, 'MONTH YYYY')* genera el campo Month en el reporte.

- En Microsoft Access, la expresión *month(OrdDate)* genera el campo Month en el reporte. En Oracle, la expresión *to\_number(to\_char(OrdDate, 'MM'))* genera el campo Month en el reporte.

Reporte de resumen de pedidos

Zip Code	Month	Order Line Count	Order Amount Sum
80111	January 2007	10	\$1149
	February 2007	21	\$2050
	Summary of 80111	31	\$3199
80113	January 2007	15	\$1541
	February 2007	11	\$1450
	Summary of 80113	31	\$2191

30. Revise el Reporte de resumen de pedidos para incluir el número de pedidos y la cantidad promedio de los pedidos, en vez de Order Line Count y Order Amount Sum. El reporte ya revisado aparece a continuación. Va a tener que usar un enunciado SELECT en la cláusula FROM o escribir dos enunciados con el fin de producir los datos para las líneas de detalle.

Reporte de resumen de pedidos

Zip Code	Month	Order Count	Average Order Amount
80111	January 2007	5	\$287.25
	February 2007	10	\$205.00
	Summary of 80111	15	\$213.27
80113	January 2007	5	\$308.20
	February 2007	4	\$362.50
	Summary of 80113	9	\$243.44

31. Para el reporte de detalle de compras, escriba un enunciado SELECT con el fin de producir los datos para las líneas de detalle. La columna de agrupamiento en el reporte es *PurchNo*. El reporte debe incluir los pedidos para el proveedor número S5095332 en febrero de 2007.

Reporte de detalle de compras

Purch Number	Purch Date	Product No	Qty	Cost	Amount
P2345877	2/11/2007	P1441567	1	\$11.99	\$11.99
		P0036577	2	\$229.00	\$458.00
	Total Purchase Amount				\$469.99
P4714645	2/10/2007	P9995676	2	\$69.00	\$138.00
		P0036566	1	\$309.00	\$309.00
	Total Purchase Amount				\$447.00

32. Para la muestra del reporte de resumen de compras, escriba un enunciado SELECT con el fin de producir los datos para las líneas de detalle. El campo Area Code del reporte lo constituyen del segundo al cuarto carácter de la columna *SuppPhone*. El campo de agrupamiento en el reporte lo constituyen del segundo al cuarto carácter de la columna *SuppPhone*. El campo Purchase Amount Sum en el reporte es la suma del resultado de la cantidad por el precio del producto. Limite el reporte a los pedidos hechos en el año 2007. También debe incluir el número de mes en el enunciado SELECT, de modo que el reporte se pueda clasificar por número de mes en lugar del campo Month. Use las siguientes expresiones para derivar las columnas calculadas que se incluyen en el reporte:

- En Microsoft Access, la expresión *mid(SuppPhone, 2, 3)* genera el campo Area Code en el reporte. En Oracle, la expresión *substr(SuppPhone, 2, 3)* genera el campo Area Code en el reporte.
- En Microsoft Access, la expresión *format(PurchDate, "mmmm yyyy")* genera el campo Month en el reporte. En Oracle, la expresión *to\_char(PurchDate, 'MONTH YYYY')* genera el campo Month en el reporte.

- En Microsoft Access, la expresión *month(PurchDate)* genera el campo Month en el reporte. En Oracle, la expresión *to\_number(to\_char(PurchDate, 'MM'))* genera el campo Month en el reporte.

**Reporte de resumen de compras**

<b>Area Code</b>	<b>Month</b>	<b>Purch Line Count</b>	<b>Purch Amount Sum</b>
303	January 2007	20	\$1 149
	February 2007	11	\$2050
Summary of 303		31	\$3199
720	January 2007	19	\$1 541
	February 2007	11	\$1 450
Summary of 720		30	\$2191

33. Revise el reporte de resumen de compras para incluir el número de compras y la cantidad promedio de ellas, en lugar de Purchase Line Count y Purchase Amount Sum. El reporte ya revisado aparece a continuación. Va a tener que usar el enunciado SELECT en la cláusula FROM o escribir dos enunciados con el fin de producir los datos para las líneas de detalle.

**Reporte de resumen de compras**

<b>Area Code</b>	<b>Month</b>	<b>Purchase Count</b>	<b>Average Purchase Amount</b>
303	January 2007	8	\$300.00
	February 2007	12	\$506.50
Summary of 303		20	\$403.25
720	January 2007	6	\$308.20
	February 2007	3	\$362.50
Summary of 303		9	\$243.44

34. Defina una vista que contenga las compras a los proveedores Connex o Cybercx. Incluya en la vista todas las columnas *Purchase*.
35. Defina una vista que contenga los detalles de las compras hechas en febrero de 2007. Incluya en la vista todas las columnas *Purchase*, *PurchLine.PurchQTY*, *PurchLine.PurchUnitCost*, así como el nombre del producto.
36. Defina una vista que contenga el número de producto, nombre, precio y cantidad disponible, además de la suma de la cantidad comprada y del costo de compra (costo unitario por cantidad comprada).
37. Utilizando la vista definida en el problema 34, escriba una consulta para incluir las compras hechas con el método de pago PO. Incluya en el resultado todas las columnas de la vista.
38. Utilizando la vista definida en el problema 35, escriba una consulta para incluir las filas que contengan la palabra Printer en el nombre de producto. Incluya en el resultado todas las columnas de la vista.
39. Utilizando la vista definida en el problema 36, escriba una consulta para incluir los productos en los que el costo de compra total sea mayor que 1 000 dólares. Incluya en el resultado el nombre del producto y el costo total de la compra.
40. Modifique la consulta en el problema 37 de modo que utilice sólo tablas base.
41. Modifique la consulta en el problema 38 de modo que utilice sólo tablas base.
42. Modifique la consulta en el problema 39 de modo que utilice sólo tablas base.

## Referencias para ampliar su estudio

Los sitios DBAZine ([www.dbazine.com](http://www.dbazine.com)) y DevX.com Database Zone ([www.devx.com](http://www.devx.com)) ofrecen numerosos consejos prácticos acerca de la formulación de consultas, SQL y desarrollo de aplicaciones de bases de datos. Por su parte, el sitio Advisor.com ([www.advisor.com/](http://www.advisor.com/)) ofrece consejos específicos sobre productos SQL: diarios técnicos para Microsoft SQL Server y Microsoft Access. La documentación de Oracle se puede encontrar en el sitio Oracle Technet ([www.oracle.com/technology](http://www.oracle.com/technology)). En el capítulo 10, Date (2003) ofrece detalles adicionales sobre problemas de actualización de las vistas relacionados con vistas de tablas múltiples. Melton y Simon (2001) describen las especificaciones sobre las consultas que se pueden actualizar en SQL:1999.

## Apéndice 10.A

### Resumen de sintaxis de SQL:2003

Este apéndice resume la sintaxis de SQL:2003 para el enunciado CREATE VIEW que se presenta en el capítulo 10 y un sencillo enunciado guía (DROP VIEW). Las convenciones utilizadas en la sintaxis son idénticas a aquellas que se usan al final del capítulo 3.

#### Enunciado CREATE VIEW

```
CREATE VIEW ViewName [ ( ColumnName* ) ]
    AS <Select-Statement>
    [ WITH CHECK OPTION ]
<Select-Statement>: -- defined in Chapter 4 and extended in Chapter 9
```

#### Enunciado DROP VIEW

```
DROP VIEW ViewName [ { CASCADE | RESTRICT } ]
-- CASCADE deletes the view and any views that use its definition.
-- RESTRICT means that the view is not deleted if any views use its definition.
```

## Apéndice 10.B

### Reglas para vistas enlazadas que se pueden actualizar en Oracle

En versiones recientes de Oracle (9i y 10g), una vista enlazada (*join view*) contiene una o más tablas o vistas en la cláusula FROM que la define. El concepto de tabla que conserva las llaves es fundamental para las vistas enlazadas que es posible actualizar. Una vista enlazada conserva una tabla si todas las llaves candidatas de la tabla pueden ser llaves candidatas para la tabla enlazada resultante. Este enunciado significa que las filas de una vista enlazada que se logre actualizar se pueden diagramar en la forma 1-1 con cada llave conservada en la tabla. En un enlace que comprende una relación 1-M, la tabla hija suele ser una llave conservada porque cada fila hija está relacionada al menos con una fila madre.

Utilizando la definición de tabla que conserva las llaves, una vista enlazada se puede actualizar si satisface las siguientes condiciones:

- No contiene la palabra clave DISTINCT, la cláusula GROUP BY, funciones conjuntas ni operaciones de conjunto (UNIONS, MINUS e INTERSECT).
- Contiene por lo menos una tabla que conserva las llaves.
- El enunciado CREATE VIEW no contiene WITH CHECK OPTION.

Una vista enlazada que sea posible actualizar ofrece soporte para las operaciones insertar, actualizar y eliminar en una tabla subyacente por enunciado de manipulación. La tabla que se puede actualizar es la tabla que conserva las llaves. Un enunciado UPDATE puede modificar (en la cláusula SET) sólo las columnas de una tabla que conserva las llaves. Un enunciado INSERT suele agregar valores para las columnas de una tabla que conserva las llaves. Un enunciado INSERT no puede contener columnas de tablas que no conserven las llaves. Las filas se llegarán a eliminar siempre y cuando la vista enlazada contenga sólo una tabla que conserve las llaves. Las vistas enlazadas con más de una tabla que conserve las llaves no ofrecen soporte para los enunciados DELETE.



# Capítulo 11

---

## Procedimientos almacenados y disparadores

### Objetivos de aprendizaje

Este capítulo explica los aspectos de motivación y diseño de los procedimientos almacenados y disparadores (*triggers*), al tiempo que le permite practicar su escritura utilizando PL/SQL, el lenguaje de programación de bases de datos de Oracle. Al finalizar este capítulo, el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Explicar las razones para escribir procedimientos almacenados y disparadores.
- Entender los aspectos de diseño del estilo de lenguaje, unión, conexión de bases de datos y procesamiento de resultados para los lenguajes de programación de bases de datos.
- Escribir procedimientos PL/SQL.
- Entender la clasificación de los disparadores.
- Escribir disparadores PL/SQL.
- Entender los procedimientos de ejecución de disparadores.

### Panorama general

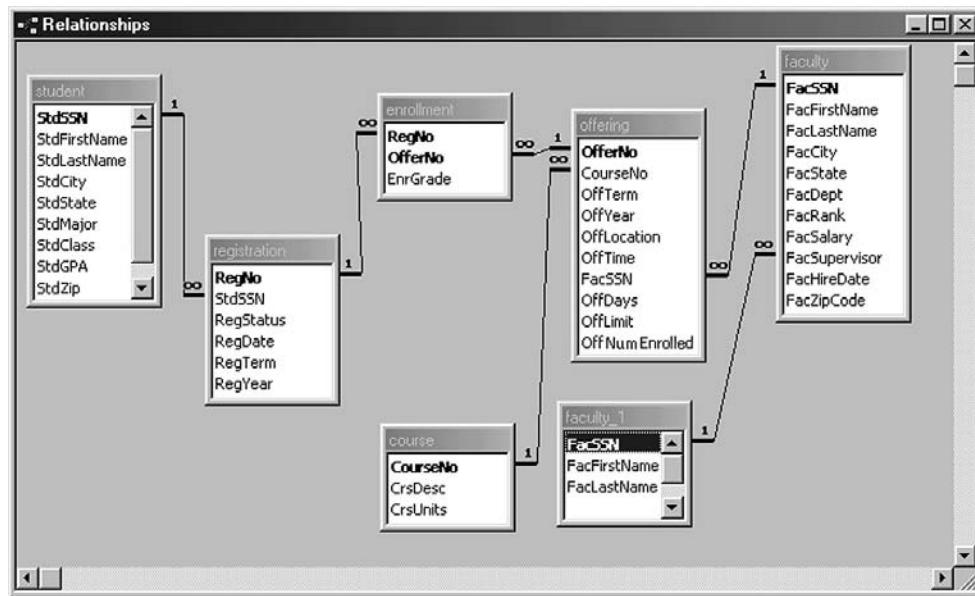
---

En el capítulo 10 usted estudió los detalles sobre el desarrollo de aplicaciones con vistas. Aprendió a definir vistas para usuario, a actualizar tablas base con vistas y a usar vistas en formularios y reportes. Este capítulo aumenta sus habilidades en el desarrollo de aplicaciones de bases de datos con procedimientos almacenados y disparadores (*triggers*). Los procedimientos almacenados ofrecen la posibilidad de volver a utilizar el código común, mientras que los disparadores proporcionan un procesamiento de reglas para las tareas comunes. Juntos, los procedimientos almacenados y los disparadores ofrecen soporte para personalizar las aplicaciones de bases de datos y mejorar la productividad al desarrollar este tipo de aplicaciones.

Con el fin de obtener habilidad en el desarrollo de aplicaciones y administración de bases de datos, es necesario entender los procedimientos almacenados y los disparadores. Como ambos se codifican en un lenguaje de programación de bases de datos, este capítulo aborda primero los antecedentes de los aspectos de motivación y diseño para los lenguajes de programación de bases de datos, así como detalles específicos acerca de PL/SQL, el lenguaje propietario de programación de bases de datos de Oracle.

Después de presentar los antecedentes sobre los lenguajes de programación de bases de datos y PL/SQL, el capítulo presenta los procedimientos almacenados y los disparadores. En

**FIGURA 11.1**  
Ventana de relación para la base de datos revisada de la universidad



cuanto a los primeros, aprenderá sobre las prácticas de motivación y codificación para procedimientos sencillos y más avanzados. Sobre los disparadores, aprenderá su clasificación, procedimientos de ejecución de disparadores y prácticas de codificación.

La presentación de detalles de PL/SQL en las secciones 11.1 a 11.3 supone que usted ya tomó un curso previo de programación de computadoras utilizando un lenguaje de programación de negocios, como Visual Basic, COBOL o Java. Si desea un estudio más amplio del material sin los detalles de programación de computadoras, antes de empezar a estudiar la sección 11.2.1 tendrá que leer las secciones 11.1.1, 11.1.2, 11.3.1, y el material de introducción en la sección 11.2. Además, los ejemplos de disparadores en la sección 11.3.2 comprenden sobre todo sentencias SQL, de modo que podrá entender los disparadores sin tener un conocimiento detallado de los enunciados PL/SQL.

Con el fin de mantener la continuidad, todos los ejemplos sobre los procedimientos almacenados y los disparadores utilizan la base de datos de la universidad revisada en el capítulo 10. La figura 11.1 muestra la ventana de relación en Access para la base de datos revisada de la universidad con el fin de facilitar su consulta.

## 11.1 Lenguajes de programación de bases de datos y PL/SQL

### **lenguaje de programación de base de datos**

lenguaje procedural con una interfaz para uno o más DBMS. La interfaz permite a un programa combinar sentencias de procedimiento con acceso a bases de datos que no son procedurales.

Después de aprender sobre el poder de las herramientas de desarrollo de aplicaciones y acceso no procedural, tal vez considere que los lenguajes de procedimientos no son necesarios para el desarrollo de aplicaciones de bases de datos. Sin embargo, a pesar de su poder, estas herramientas no son soluciones completas para el desarrollo de aplicaciones de bases de datos comerciales. Esta sección presenta los motivos y los aspectos de diseño para los lenguajes de programación de bases de datos, y los detalles acerca de PL/SQL, el lenguaje de programación de bases de datos de Oracle.

### **11.1.1 Motivación para los lenguajes de programación de base de datos**

Un lenguaje de programación de base de datos es un lenguaje procedural con una interfaz para uno o más DBMS. La interfaz permite que un programa combine sentencias de procedimiento con el acceso a la base de datos, que por lo regular es no procedural. Esta subsección estudia tres motivaciones primarias (personalización, procesamiento por lote y operaciones complejas) para

usar un lenguaje de programación de bases de datos, así como dos motivaciones secundarias (eficiencia y portabilidad).

### *Personalización*

La mayor parte de las herramientas para desarrollo de aplicaciones de base de datos ofrece soporte para la personalización. Ésta es necesaria porque ninguna herramienta proporciona una solución completa para el desarrollo de aplicaciones de bases de datos complejas. La personalización permite a una organización utilizar el poder integrado de una herramienta en combinación con el código personalizado para cambiar las acciones predeterminadas de la herramienta y agregar acciones nuevas, adicionales a las que la herramienta soporta.

Con el fin de ofrecer soporte para el código personalizado, la mayoría de las herramientas para el desarrollo de aplicaciones de base de datos utilizan código basado en los eventos. En este estilo de código, un evento dispara (*trigger*) la ejecución de un procedimiento. Un modelo de eventos incluye eventos para las acciones del usuario, como hacer clic en un botón, así como eventos internos, como antes de que el registro de la base de datos sea actualizado. Un procedimiento de evento puede tener acceso a los valores de los controles en formularios y reportes, así como recuperar y actualizar filas de la base de datos. Los procedimientos de eventos se codifican utilizando un lenguaje de programación de base de datos, que a menudo es un lenguaje patentado que proporciona un proveedor de DBMS. La codificación de eventos es muy común en el desarrollo de aplicaciones comerciales.

### *Procesamiento por lote (batch)*

A pesar del crecimiento del procesamiento de base de datos en línea, el procesamiento por lote sigue siendo una forma importante de procesar el trabajo en una base de datos. Por ejemplo, el procesamiento de cheques casi siempre es un proceso por lote en el que un banco procesa grandes grupos o lotes de cheques en horas en que no hay mucho trabajo. Por lo regular, el procesamiento por lote comprende una demora entre el momento en que un evento se presenta y su captura en una base de datos. En el caso del procesamiento de cheques, éstos se presentan para su pago a un comerciante, pero el banco los procesa más adelante. Algunas aplicaciones de procesamiento por lote, como la preparación de estados de cuenta, comprenden un tiempo de cierre en lugar de una demora. El procesamiento en lote en situaciones que comprenden demoras y cortes de tiempo proporciona economías a escala significativas para compensar la desventaja de la información fuera de tiempo. Aun con el crecimiento continuo del comercio en la web, el procesamiento por lote seguirá siendo un método importante para procesar el trabajo en bases de datos.

El desarrollo de aplicaciones para el procesamiento por lote implica escribir programas para computadora en un lenguaje de programación de bases de datos. Como pocas herramientas de desarrollo ofrecen soporte para el procesamiento en lote, la codificación puede ser detallada y laboriosa. Por lo general, el programador debe escribir el código para leer los datos de entrada, manipular la base de datos y crear registros de salida para mostrar el resultado del procesamiento.

### *Operaciones complejas*

Por definición, el acceso no procedural a bases de datos no soporta todas las recuperaciones posibles de una base de datos. El diseño de un lenguaje no procedural representa un sacrificio entre la cantidad de código y la complejidad computacional. Para permitir el cómputo de uso general es necesario un lenguaje procedural. Para reducir la codificación, los lenguajes procedurales ofrecen soporte para la especificación compacta de operaciones importantes y comunes. La sentencia SELECT de SQL soporta las operaciones de álgebra relacional, además de ordenar y agrupar. Para llevar a cabo recuperaciones de bases de datos más allá de las operaciones de álgebra relacional, es necesario codificar en un lenguaje de programación de base de datos.

La clausura transitiva es una operación importante que no tiene soporte en la mayoría de las implementaciones de SQL. Esta operación es importante para las consultas que comprenden relaciones de autoreferencia. Por ejemplo, el operador de clausura transitiva es necesario para recuperar todos los empleados administrados directa o indirectamente utilizando una relación autoreferenciada. Esta operación comprende operaciones de autounión, pero el número de ellas depende de la profundidad (número de capas de subordinados) en un organigrama.

Aunque la cláusula WITH RECURSIVE para las operaciones de clausura transitiva se introdujo en SQL:1999, la mayoría de los DBMS no la han implementado. En casi todos los DBMS, es preciso codificar las operaciones de clausura transitiva utilizando un lenguaje de programación de base de datos. Para codificar una operación de clausura transitiva, es necesario ejecutar una consulta de autounión de manera repetitiva dentro de un ciclo, hasta que la consulta dé un resultado vacío.

#### *Otras motivaciones*

La eficiencia y portabilidad son dos razones más para usar un lenguaje de programación de base de datos. Cuando la desconfianza en la optimización de los compiladores de bases de datos era muy alta (hasta mediados de la década de 1990), la eficiencia era una motivación primaria para usar un lenguaje de programación de base de datos. Con el fin de evitar la optimización de los compiladores, algunos proveedores de DBMS ofrecieron soporte para el acceso de un registro a la vez, en el que el programador determinaba el plan de acceso para las consultas complejas. Conforme aumentó la confianza en los compiladores de base de datos, la eficiencia se volvió menos importante. Sin embargo, con las aplicaciones web complejas y las inmaduras herramientas de desarrollo para la misma, la eficiencia se ha convertido en un aspecto importante en algunas aplicaciones. Conforme las herramientas de desarrollo para la web maduren, la eficiencia será menos importante.

La portabilidad puede ser importante en algunos entornos. La mayor parte de las herramientas para el desarrollo de aplicaciones y los lenguajes de programación de base de datos son propietarias. Si una organización quiere seguir siendo neutral con los proveedores, puede crear una aplicación utilizando un lenguaje de programación no propietario (como Java), además de una interfaz de base de datos estándar. Si lo que se busca es la neutralidad del DBMS (y no la de una herramienta para el desarrollo de aplicaciones), algunas herramientas permiten la conexión con diversos DBMS a través de interfaces estándar, como Open Database Connectivity (ODBC) y Java Database Connectivity (JDBC). La portabilidad es una preocupación sobre todo para el acceso a bases de datos en la web, donde una aplicación debe ser compatible con muchos tipos de servidores y visualizadores.

### **11.1.2 Aspectos de diseño**

Antes de iniciar el estudio de cualquier lenguaje de programación de base de datos, es preciso entender los aspectos de diseño relacionados con la integración de un lenguaje procedural con un lenguaje no procedural. El hecho de entender estos aspectos le ayudará a diferenciar entre los distintos lenguajes que existen en el mercado y a comprender las características de un lenguaje específico. Por lo regular, cada DBMS proporciona varias alternativas para lenguajes de programación de bases de datos. Esta sección estudia los aspectos de diseño del estilo del lenguaje, unión, conexión de base de datos y procesamiento de resultados, poniendo énfasis en las opciones de diseño que se especificaron por primera vez en SQL:1999 y se refinaron en SQL:2003. Muchos proveedores de DBMS se adaptaron a las especificaciones de SQL:2003.

#### *Estilo de lenguaje*

SQL:2003 ofrece dos estilos de lenguaje para integrar un lenguaje procedural con SQL. Una interfaz en el nivel de sentencias comprende cambios en la sintaxis del lenguaje de programación del anfitrión para aceptar las sentencias integradas de SQL. El lenguaje del anfitrión contiene sentencias adicionales para establecer conexiones de base de datos, ejecutar sentencias SQL, utilizar los resultados de una sentencia SQL, asociar las variables de programación con las columnas de la base de datos, manejar las excepciones en las sentencias SQL y manipular los descriptores de base de datos. Hay interfaces en el nivel de sentencias para lenguajes estándar y propietarios. Para los lenguajes estándar como C, Java y COBOL, algunos DBMS ofrecen un precompilador para procesar las sentencias antes de invocar el lenguaje de programación del compilador. La mayoría de los DBMS ofrecen también lenguajes propietarios como el lenguaje PL/SQL de Oracle, con una interfaz en el nivel de sentencias para soportar el SQL integrado.

La especificación SQL:2003 define el lenguaje Persistent Stored Modules (SQL/PSM) como un lenguaje de programación de base de datos. Como el SQL/PSM se definió después

#### **interfaz en el nivel de sentencias (statement-level)**

estilo de lenguaje para integrar un lenguaje de programación con un lenguaje no procedural, como SQL. Una interfaz en el nivel de sentencias comprende cambios en la sintaxis del lenguaje de programación del anfitrión para aceptar las sentencias SQL integrados.

de que muchos proveedores de DBMS ya habían utilizado los lenguajes propietarios en forma generalizada, prácticamente ningún proveedor cumple con las especificaciones del estándar SQL/PSM. Sin embargo, dicho estándar ha influido en el diseño de lenguajes de programación de bases de datos propietarios, como Oracle PL/SQL.

**interfaz en el nivel de llamadas (CLI)** estilo de lenguaje para integrar un lenguaje de programación con un lenguaje no procedural, como SQL. Una CLI incluye una serie de procedimientos y definiciones de tipos para manipular los resultados de las sentencias SQL en los programas de computación.

El segundo estilo de lenguaje que proporciona SQL:2003 se conoce como interfaz en el nivel de llamadas (CLI, call-level interface). El CLI de SQL:2003 contiene una serie de procedimientos y definiciones de tipos para datos de SQL. Los procedimientos ofrecen una funcionalidad similar a las sentencias adicionales en una interfaz de nivel de sentencias. La CLI de SQL:2003 es más difícil de aprender y de usar que una interfaz en el nivel de sentencias. Sin embargo, la CLI es portátil entre los lenguajes anfitriones, mientras que la interfaz en el nivel de sentencias no es portátil y no tiene soporte en todos los lenguajes de programación.

Las interfaces en el nivel de llamadas que se usan con mayor frecuencia son Open Database Connectivity (ODBC), que tiene soporte en Microsoft, y Java Database Connectivity (JDBC), con soporte en Oracle. Tanto Microsoft como Oracle han cooperado en los esfuerzos de los estándares SQL, por lo que las versiones más recientes de estas CLI propietarias son compatibles con la CLI de SQL:2003. Gracias a las bases establecidas por los usuarios es probable que estas interfaces se sigan usando con más frecuencia que la CLI de SQL:2003.

### *Unión*

Para un lenguaje de programación de base de datos, la unión comprende la asociación de una sentencia SQL con su plan de acceso. Recuerde que en el capítulo 8 estudiamos que un compilador SQL determina el mejor plan de acceso para una sentencia SQL después de la búsqueda detallada de los planes de acceso posibles. La unión estática comprende la determinación del plan de acceso en el momento de compilar. Como el proceso de optimización puede consumir gran cantidad de recursos de cómputo, lo mejor es determinar el plan de acceso al momento de compilar y luego volver a usarlo para sentencias ejecutadas varias veces. Sin embargo, en algunas aplicaciones no es posible predeterminar los datos a recuperar. En estas situaciones es necesaria la unión dinámica, en la que el plan de acceso para una sentencia se determina cuando se ejecuta durante el tiempo de ejecución de una aplicación. Aun en estas situaciones dinámicas, resulta útil volver a utilizar el plan de acceso para una sentencia, si la aplicación lo ejecuta repetidamente.

SQL:2003 especifica tanto la unión estática como la dinámica para soportar el rango de las aplicaciones de base de datos. Una interfaz en el nivel de la sentencia ofrece soporte para uniones estáticas y dinámicas. Las sentencias SQL integradas tienen enlace estática. Las sentencias SQL dinámicas tienen soporte en la sentencia EXECUTE de SQL:2003 que contiene una sentencia SQL como parámetro de entrada. Si una aplicación ejecuta varias veces una sentencia dinámica, la sentencia PREPARE de SQL:2003 apoya el uso del plan de acceso. La CLI de SQL:2003 ofrece el procedimiento Prepare() para volver a usar el plan de acceso.

### *Conexión de base de datos*

Una conexión de base de datos identifica la base de datos que utiliza una aplicación. Esta conexión puede ser implícita o explícita. Para los procedimientos y disparadores almacenados en una base de datos, la conexión es implícita. Las sentencias SQL en los disparadores y procedimientos tienen acceso implícito a la base de datos que contiene los disparadores y procedimientos.

En los programas externos a una base de datos, la conexión es explícita. SQL:2003 especifica la sentencia CONNECT y otras sentencias relacionadas para interfaces en el nivel de sentencia, así como el procedimiento Connect() y procedimientos relacionados en la CLI. Una base de datos se identifica mediante una dirección web o un identificador de bases de datos que contiene una dirección web. El uso de un identificador de bases de datos evita al programador la necesidad de conocer las direcciones web específicas para una base de datos, además de que ofrece al administrador de servidores mayor flexibilidad para reubicar una base de datos en un lugar diferente del servidor.

### *Procesamiento de resultados*

Para procesar los resultados de sentencias SQL, los lenguajes de programación de base de datos deben resolver las diferencias en los tipos de datos y la orientación del procesamiento. Los tipos

de datos en un lenguaje de programación pueden no corresponder exactamente con los tipos de datos SQL estándar. Para solucionar este problema, la interfaz de la base de datos proporciona sentencias o procedimientos para diagramar entre los tipos de datos del lenguaje de programación y los tipos de datos SQL.

El resultado de una sentencia SELECT puede ser una fila o un conjunto de filas. Para las sentencias SELECT que regresan cuando mucho una fila (por ejemplo, recuperación mediante llave primaria), la especificación SQL:2003 permite que los valores del resultado se almacenen en las variables del programa. En la interfaz en el nivel de sentencia, SQL:2003 proporciona la cláusula USING para guardar los valores del resultado en las variables del programa. La CLI de SQL:2003 ofrece valores para el almacenamiento implícito del resultado utilizando los registros de descriptores previamente definidos a los que se puede tener acceso en un programa.

### **cursor**

una construcción en un lenguaje de programación de base de datos que permite el almacenamiento y la iteración de un conjunto de registros obtenidos por una sentencia SELECT. Un cursor es similar a un conjunto dinámico en el que el tamaño de este último se determina por el tamaño del resultado de la consulta.

Un cursor debe usarse para las sentencias SELECT que regresen más de una fila. Un cursor permite el almacenamiento y la iteración de un conjunto de registros obtenidos mediante una sentencia SELECT. Un cursor es similar a un conjunto dinámico en el que el tamaño de éste se determina por el tamaño del resultado de la consulta. Para las interfaces en el nivel de la sentencia, SQL:2003 ofrece sentencias para declarar cursos, cursos abiertos y cerrados, posición de los cursos y obtención de valores de los mismos. La CLI de SQL:2003 ofrece procedimientos con la misma funcionalidad que la interfase en el nivel de la sentencia. La sección 11.2.3 ofrece detalles sobre los cursos para PL/SQL.

### **11.1.3 Sentencias PL/SQL**

Programming Language/Structured Query Language (PL/SQL) es un lenguaje de programación de bases de datos propietario para el DBMS de Oracle. Desde su introducción en 1992, Oracle ha agregado características en forma continua, de modo que tiene las características de un lenguaje de programación moderno, así como una interfaz en el nivel de sentencia para SQL. Como el PL/SQL es un lenguaje que los desarrolladores de Oracle utilizan con frecuencia y Oracle es un DBMS empresarial que se usa a menudo, este capítulo emplea el PL/SQL para ilustrar los procedimientos almacenados y los disparadores.

Con el fin de prepararlo para que lea y codifique procedimientos almacenados y disparadores, esta sección presenta ejemplos de sentencias PL/SQL. Después de leer esta sección, usted deberá entender la estructura de las sentencias PL/SQL y podrá escribir sentencias PL/SQL utilizando como lineamientos las sentencias de los ejemplos. Esta sección muestra suficientes ejemplos de la sentencia PL/SQL para que, al terminar el capítulo, pueda leer y escribir procedimientos almacenados y disparadores de mediana complejidad. Sin embargo, no ilustra todos las sentencias PL/SQL ni todas sus variaciones.

Esta sección no es un tutorial de programación de computadoras. Para seguir el resto del capítulo, es preciso que haya tomado un curso de programación previo o que tenga una experiencia equivalente. Usted encontrará que las sentencias PL/SQL son similares a los que aparecen en otros lenguajes de programación modernos como C, Java y Visual Basic.

#### *Fundamentos de PL/SQL*

Las sentencias PL/SQL contienen palabras y símbolos reservados, identificadores de usuario y valores constantes. Las palabras reservadas en PL/SQL no distinguen entre mayúsculas y minúsculas. Los símbolos reservados incluyen el punto y coma (;) para finalizar las sentencias, además de operadores como + y -. Los identificadores de usuario dan nombres a las variables, constantes y otras construcciones PL/SQL. Al igual que las palabras reservadas, los identificadores de usuario no distinguen entre mayúsculas y minúsculas. La siguiente lista define sus restricciones:

- Deben tener un máximo de 30 caracteres.
- Deben empezar con una letra.
- Los caracteres permitidos son letras (mayúsculas o minúsculas), números, el signo de pesos, el símbolo de gato (#), y el subrayado.
- No deben ser iguales a ningún símbolo o palabra reservada.
- No deben ser iguales a otros identificadores, nombres de tablas o nombres de columnas.

Una sentencia PL/SQL puede contener valores constantes para números y cadenas de caracteres en ciertas palabras reservadas. La siguiente lista ofrece antecedentes acerca de las constantes PL/SQL:

- Las constantes numéricas pueden ser números enteros (100), números con punto decimal (1.67), números negativos (-150.15) y números en notaciones científicas (3.14E7).
- Las cadenas constantes están encerradas entre comillas sencillas como 'ésta es una cadena'. No use comillas sencillas para encerrar constantes numéricas o booleanas. Las cadenas constantes distinguen entre mayúsculas y minúsculas, de modo que 'Ésta es una cadena' es un valor diferente a 'ésta es una cadena'. Si necesita utilizar comillas sencillas como parte de una cadena, tendrá que escribir dos comillas, como 'today's date'.
- Las constantes booleanas son las palabras reservadas TRUE y FALSE.
- La palabra reservada NULL se puede usar como número, cadena o constante booleana. Para las cadenas, dos comillas sencillas '' sin nada en medio indican el valor NULL.
- PL/SQL no proporciona constantes de fecha. Tiene que usar la función **To\_Date** para convertir una cadena constante en un valor de fecha.

#### *Declaración de variables y sentencias de asignación*

La declaración de variable contiene un nombre de variable (un identificador de usuario), un tipo de datos y un valor predeterminado opcional. La tabla 11.1 muestra los tipos de datos PL/SQL comunes. Además de usar los tipos predefinidos, el tipo de una variable puede ser un tipo definido por el usuario creado con una sentencia TYPE. Un valor predeterminado se puede indicar con la palabra clave DEFAULT o el símbolo de asignación (:=). La palabra clave DECLARE debe ir antes de la declaración de la primera variable, como muestra el ejemplo 11.1.

#### EJEMPLO 11.1

#### Declaración de variables PL/SQL

Las líneas que empiezan con doble guión son comentarios.

```
DECLARE
    aFixedLengthString      CHAR(6) DEFAULT 'ABCDEF';
    aVariableLengthString   VARCHAR2(30);
    anIntegerVariable       INTEGER := 0;
    aFixedPrecisionVariable DECIMAL(10, 2);
    -- Uses the SysDate function for the default value
    aDateVariable          DATE DEFAULT SysDate;
```

**TABLA 11.1**  
Resumen de tipos comunes de datos PL/SQL

Categoría	Tipos de dato	Comentarios
Cadena (string)	CHAR(L), VARCHAR2(L)	CHAR para cadenas de longitud fija, VARCHAR2 para cadenas de longitud variable; L para la longitud máxima
Numéricos	INTEGER, SMALLINT, POSITIVE, NUMBER(W,D), DECIMAL(W,D), FLOAT, REAL	W para el ancho; D para el número de dígitos a la derecha del punto decimal
Lógicos	BOOLEAN	Valores TRUE, FALSE
Fecha	DATE	Guarda información sobre la fecha y la hora, incluidos siglo, año, mes, día, hora, minuto y segundo. Una fecha ocupa 7 bytes.

Para las variables relacionadas con las columnas de una tabla de base de datos, PL/SQL ofrece declaraciones ancladas. Estas declaraciones evitan que el programador tenga que conocer los tipos de datos de las columnas de la base de datos. Una declaración anclada incluye un nombre de columna totalmente calificado, seguido por la palabra clave %TYPE. El ejemplo 11.2 muestra las declaraciones de variables ancladas que usan columnas de la base de datos de la universidad del capítulo 10. La última declaración anclada comprende una variable que utiliza el tipo relacionado con una variable definida previamente.

### EJEMPLO 11.2 Declaraciones de variable anclada PL/SQL

```
DECLARE
    anOffTerm Offering.OffTerm%TYPE;
    anOffYear Offering.OffYear%TYPE;
    aCrsUnits Course.CrsUnits%TYPE;
    aSalary1 DECIMAL(10,2);
    aSalary2 aSalary1%TYPE;
```

Oracle también ofrece tipos de datos estructurados para combinar tipos de datos primitivos. Oracle ofrece soporte para arreglos de longitud variable (VARRAY), tablas (TABLE) y registros (RECORD) para combinar tipos de datos. Si desea información sobre los tipos de datos estructurados, tendrá que consultar la documentación de Oracle en línea, como la *Guía del usuario de PL/SQL*.

Las sentencias de asignación comprenden una variable, el símbolo de asignación (:=) y una expresión a la derecha. Las expresiones pueden incluir combinaciones de constantes, variables, funciones y operadores. Al evaluarla, una expresión produce un valor. El ejemplo 11.3 muestra sentencias de asignación con diversos elementos de expresión.

### EJEMPLO 11.3 Ejemplos de asignación en PL/SQL

Se supone que las variables utilizadas en los ejemplos se declararon previamente. Las líneas que empiezan con doble guión son comentarios.

```
aFixedLengthString := 'XYZABC';
-- || is the string concatenation function
aVariableLengthString := aFixedLengthString || 'ABCDEF';
anIntegerVariable := anAge + 1;
aFixedPrecisionVariable := aSalary * 0.10;
-- To_Date is the date conversion function
aDateVariable := To_Date ('30-Jun-2006');
```

### *Sentencias condicionales*

PL/SQL ofrece las sentencias IF y CASE para la toma de decisión condicional. En una sentencia IF, la expresión lógica o condición para evaluar TRUE, FALSE o NULL se escribe después de la palabra IF. Las condiciones incluyen expresiones de comparación que usan los operadores de comparación (tabla 11.2) conectados con los operadores lógicos AND, OR y NOT. Puede usar paréntesis para aclarar el orden de la evaluación en condiciones complejas; y su uso es necesario al combinar los operadores AND y OR. Las condiciones se evalúan utilizando la lógica de tres valores que describimos en el capítulo 9 (sección 9.4).

**TABLA 11.2**  
**Lista de operadores de comparación en PL/SQL**

Operador	Significado
=	Igual a
<>	No igual a
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

Como en otros lenguajes, en PL/SQL la sentencia IF tiene diversas variaciones. El ejemplo 11.4 ilustra la primera variación conocida como sentencia IF-THEN. Es posible utilizar cualquier cantidad de variaciones entre las palabras clave THEN y END IF. El ejemplo 11.5 muestra la segunda variación conocida como sentencia IF-THEN-ELSE. Esta sentencia permite un conjunto de sentencias alternativas si la condición es falsa. La tercera variación (IF-THEN-ELSIF), que ilustra el ejemplo 11.6, permite una condición para cada cláusula ELSIF, con una cláusula final ELSE si todas las condiciones son falsas.

**IF-THEN Statement:**

```
IF condition THEN
    sequence of statements
END IF;
```

**IF-THEN-ELSE Statement:**

```
IF condition THEN
    sequence of statements 1
ELSE
    sequence of statements 2
END IF;
```

**IF-THEN-ELSIF Statement:**

```
IF condition1 THEN
    sequence of statements 1
ELSIF condition2 THEN
    sequence of statements 2
ELSIF conditionN THEN
    sequence of statements N
ELSE
    sequence of statements N+1
END IF;
```

**EJEMPLO 11.4**

**Ejemplos de la sentencia IF-THEN**

Se supone que las variables utilizadas en los ejemplos se declararon previamente.

```
IF aCrsUnits > 3 THEN
    CourseFee := BaseFee + aCrsUnits * VarFee;
END IF;

IF anOffLimit > NumEnrolled OR CourseOverRide = TRUE THEN
    NumEnrolled := NumEnrolled + 1;
    EnrDate := SysDate;
END IF;
```

**EJEMPLO 11.5 Ejemplos de la sentencia IF-THEN-ELSE**

Se supone que las variables utilizadas en los ejemplos se declararon previamente.

```

IF aCrsUnits > 3 THEN
    CourseFee := BaseFee + ((aCrsUnits - 3) * VarFee);
ELSE
    CourseFee := BaseFee;
END IF;

IF anOffLimit > NumEnrolled OR CourseOverRide = TRUE THEN
    NumEnrolled := NumEnrolled + 1;
    EnrDate := SysDate;
ELSE
    Enrolled := FALSE;
END IF;
```

**EJEMPLO 11.6 Ejemplos de la sentencia IF-THEN-ELSIF**

Se supone que las variables utilizadas en los ejemplos se declararon previamente.

```

IF anOffTerm = 'Fall' AND Enrolled := TRUE THEN
    FallEnrolled := FallEnrolled + 1;
ELSIF anOffTerm = 'Spring' AND Enrolled := TRUE THEN
    SpringEnrolled := SpringEnrolled + 1;
ELSE
    SummerEnrolled := SummerEnrolled + 1;
END IF;

IF aStdClass = 'FR' THEN
    NumFR := NumFR + 1;
    NumStudents := NumStudents + 1;
ELSIF aStdClass = 'SO' THEN
    NumSO := NumSO + 1;
    NumStudents := NumStudents + 1;
ELSIF aStdClass = 'JR' THEN
    NumJR := NumJR + 1;
    NumStudents := NumStudents + 1;
ELSIF aStdClass = 'SR' THEN
    NumSR := NumSR + 1;
    NumStudents := NumStudents + 1;
END IF;
```

La sentencia CASE utiliza un selector en lugar de una condición. Un selector es una expresión cuyo valor determina una decisión. El ejemplo 11.7 muestra la sentencia CASE que corresponde a la segunda parte del ejemplo 11.6. La sentencia CASE se usó por primera vez en PL/SQL para Oracle 9i. Las versiones anteriores de Oracle dan un error de sintaxis para el ejemplo 11.7.

**EJEMPLO 11.7****Ejemplo de la sentencia CASE que corresponde a la segunda parte del ejemplo 11.6**

Se supone que las variables utilizadas en el ejemplo se declararon previamente.

```
CASE aStdClass
    WHEN 'FR' THEN
        NumFR := NumFR + 1;
        NumStudents := NumStudents + 1;
    WHEN 'SO' THEN
        NumSO := NumSO + 1;
        NumStudents := NumStudents + 1;
    WHEN 'JR' THEN
        NumJR := NumJR + 1;
        NumStudents := NumStudents + 1;
    WHEN 'SR' THEN
        NumSR := NumSR + 1;
        NumStudents := NumStudents + 1;
END CASE;
```

**CASE Statement:**

```
CASE selector
    WHEN expression1 THEN sequence of statements 1
    WHEN expression2 THEN sequence of statements 2
    WHEN expressionN THEN sequence of statements N
    [ ELSE sequence of statements N+1 ]
END CASE;
```

*Sentencias de iteración*

PL/SQL contiene tres sentencias de iteración, además de una sentencia para finalizar un ciclo. La sentencia FOR LOOP se itera en un rango de valores enteros, como muestra el ejemplo 11.8. La sentencia WHILE LOOP se itera hasta que una condición de paro sea falsa, como ilustra el ejemplo 11.9. La sentencia LOOP se itera hasta que la sentencia EXIT finaliza la terminación, como muestra el ejemplo 11.10. Debemos señalar que la sentencia EXIT también se puede usar en las sentencias FOR LOOP y WHILE LOOP para provocar la terminación temprana de un ciclo.

**EJEMPLO 11.8****Ejemplo de la sentencia FOR LOOP**

Se supone que las variables utilizadas en el ejemplo se declararon previamente.

```
FOR Idx IN 1 . . NumStudents LOOP
    TotalUnits := TotalUnits + (Idx * aCrsUnits);
END LOOP;
```

**EJEMPLO 11.9****Sentencia WHILE LOOP correspondiente al ejemplo 11.8**

```
Idx := 1;
WHILE Idx <= NumStudents LOOP
    TotalUnits := TotalUnits + (Idx * aCrsUnits);
    Idx := Idx + 1;
END LOOP;
```

**EJEMPLO 11.10 Sentencia LOOP correspondiente al ejemplo 11.8**

```

Idx := 1;
LOOP
    TotalUnits := TotalUnits + (Idx * aCrsUnits);
    Idx := Idx + 1;
    EXIT WHEN Idx > NumStudents;
END LOOP;

```

**FOR LOOP Statement:**

```

FOR variable IN BeginExpr .. EndExpr LOOP
    sequence of statements
END LOOP;

```

**WHILE LOOP Statement:**

```

WHILE condition LOOP
    sequence of statements
END LOOP;

```

**LOOP Statement:**

```

LOOP
    sequence of statements containing an EXIT statement
END LOOP;

```

**11.1.4 Ejecución de las sentencias PL/SQL en bloques anónimos**

PL/SQL es un lenguaje estructurado en bloques. En la sección 11.2 aprenderá sobre los bloques con nombres, pero esta sección presenta los bloques anónimos con el fin de que pueda ejecutar los ejemplos de sentencias en SQL \*Plus, la herramienta más popular para la ejecución de sentencias PL/SQL. Los bloques anónimos también son útiles para probar procedimientos y disparadores. Antes de presentarlos, ofrecemos una breve introducción a SQL \*Plus.

Para usar SQL \*Plus, necesita un nombre de registro en Oracle y una contraseña. La autenticación en SQL \*Plus es diferente que en su sistema operativo. Después de conectarse a SQL \*Plus, verá el apuntador SQL>. En éste puede capturar sentencias SQL, bloques PL/SQL y comandos SQL \*Plus. La tabla 11.3 presenta los comandos más comunes para SQL \*Plus. Para finalizar una sentencia individual o un comando en SQL \*Plus debe usar punto y coma al final de la sentencia o comando. Para terminar un grupo de sentencias o comandos debe utilizar una diagonal (/) sola en la línea.

**TABLA 11.3**  
**Lista de comandos comunes en SQL \*Plus**

Comando	Ejemplo y significado
CONNECT	CONNECT UserName@DatabaseId/Password abre una conexión a DatabaseId para UserName con Password
DESCRIBE	DESCRIBE TableName muestra las columnas de TableName
EXECUTE	EXECUTE Statement ejecuta el sentencia PL/SQL
HELP	HELP ColumnName describe ColumnName
SET	SET SERVEROUTPUT ON hace que se desplieguen los resultados de las sentencias PL/SQL
SHOW	SHOW ERRORS hace que se desplieguen los errores en la compilación
SPOOL	SPOOL FileName hace que el resultado se escriba en FileName. SPOOL OFF detiene la escritura en un archivo
/	Se usa sola en una línea para finalizar un conjunto de sentencias o comandos de SQL *Plus

Un bloque PL/SQL contiene una sección de declaración opcional (palabra clave DECLARE), una sección ejecutable (palabra clave BEGIN) y una sección de excepción opcional (palabra clave EXCEPTION). Esta sección presenta bloques anónimos que contienen secciones de declaración y ejecutables. La sección 11.2 ilustra la sección de excepción.

#### **Block Structure:**

```
[ DECLARE
    sequence of declarations ]
BEGIN
    sequence of statements
[ EXCEPTION
    sequence of statements to respond to exceptions ]
END;
```

Para demostrar los bloques anónimos, el ejemplo 11.11 calcula la suma y el producto de los enteros del 1 al 10. El procedimiento *Dbms\_Output.Put\_Line* muestra los resultados. El paquete *Dbms\_Output* contiene procedimientos y funciones para leer y escribir líneas en un búfer. El ejemplo 11.12 modifica el ejemplo 11.11 para calcular la suma de los números impares, así como el producto de los números pares.

---

#### **EJEMPLO 11.11 Bloque anónimo para calcular la suma y el producto**

La primera línea (comando SET) y la última línea (/) no son parte del bloque anónimo.

```
-- SQL *Plus command
SET SERVEROUTPUT ON;
-- Anonymous block
DECLARE
    TmpSum    INTEGER;
    TmpProd   INTEGER;
    Idx       INTEGER;
BEGIN
    -- Initialize temporary variables
    TmpSum := 0;
    TmpProd := 1;
    -- Use a loop to compute the sum and product
    FOR Idx IN 1 . . 10 LOOP
        TmpSum := TmpSum + Idx;
        TmpProd := TmpProd * Idx;
    END LOOP;
    -- Display the results
    Dbms_Output.Put_Line('Sum is ' || To_Char(TmpSum) );
    Dbms_Output.Put_Line('Product is ' || To_Char(TmpProd) );
END;
/
```

---

#### **EJEMPLO 11.12 Bloque anónimo para calcular la suma y el producto de los números pares**

El comando SET no es necesario si se utilizó para el ejemplo 11.11 en la misma sesión de SQL \*Plus.

```

SET SERVEROUTPUT ON;
DECLARE
    TmpSum      INTEGER;
    TmpProd     INTEGER;
    Idx         INTEGER;
BEGIN
    -- Initialize temporary variables
    TmpSum := 0;
    TmpProd := 1;
    -- Use a loop to compute the sum of the even numbers and
    -- the product of the odd numbers.
    -- Mod(X,Y) returns the integer remainder of X/Y.
    FOR Idx IN 1 .. 10 LOOP
        IF Mod(Idx,2) = 0 THEN -- even number
            TmpSum := TmpSum + Idx;
        ELSE
            TmpProd := TmpProd * Idx;
        END IF;
    END LOOP;
    -- Display the results
    Dbms_Output.Put_Line('Sum is ' || To_Char(TmpSum) );
    Dbms_Output.Put_Line('Product is ' || To_Char(TmpProd) );
END;
/

```

## 11.2 Procedimientos almacenados

Con los antecedentes acerca de los lenguajes de programación de bases de datos y PL/SQL, ahora está preparado para aprender los procedimientos almacenados. Los lenguajes de programación han ofrecido soporte para los procedimientos desde los primeros días de la computación empresarial. Los procedimientos permiten manejar la complejidad al dejar que las tareas de computación se dividan en bloques fáciles de manejar. Un procedimiento de base de datos es como un procedimiento de lenguaje de programación, sólo que se maneja mediante el DBMS y no en el ambiente de programación. La siguiente lista explica las razones para que un DBMS administre los procedimientos:

- Un DBMS puede compilar el código del lenguaje de programación con las sentencias SQL en un procedimiento almacenado. Además, un DBMS puede detectar cuándo es necesario volver a compilar las sentencias SQL en un procedimiento debido a cambios en las definiciones de la base de datos.
- Los procedimientos almacenados permiten la flexibilidad para el desarrollo cliente-servidor. Los procedimientos almacenados se guardan en un servidor y no se repiten en cada cliente. En los primeros días de la computación cliente-servidor, la capacidad de guardar procedimientos en un servidor era la principal motivación para los procedimientos almacenados. Con el desarrollo de objetos distribuidos en la web, esta motivación ya no es importante porque existen otras tecnologías para manejar los procedimientos almacenados en servidores remotos.
- Los procedimientos almacenados permiten el desarrollo de operaciones y funciones más complejas que aquellas que tienen soporte en SQL. El capítulo 18 describe la importancia de los procedimientos y funciones especializados en las bases de datos orientadas a objetos.
- Los administradores de bases de datos pueden manejar procedimientos almacenados con las mismas herramientas para manejar otras partes de una aplicación de base de datos. Lo más importante es que los procedimientos almacenados se manejan a través del sistema de seguridad del DBMS.

Esta sección abarca los procedimientos, funciones y paquetes PL/SQL. Presentamos algunas partes adicionales de PL/SQL (cursos y excepciones) para demostrar la utilidad de los procedimientos almacenados. Los *scripts* de prueba suponen que las tablas de la universidad se llenan de acuerdo con los datos del sitio web del libro de texto.

### 11.2.1 Procedimientos PL/SQL

En PL/SQL, un procedimiento es un bloque con nombre que tiene un conjunto opcional de parámetros. Cada parámetro contiene un nombre de parámetro, un uso (IN, OUT, IN OUT) y un tipo de dato. Un parámetro de entrada (IN) no debe cambiar dentro de un procedimiento. A un parámetro de salida (OUT) se le da un valor dentro de un procedimiento. Un parámetro de entrada-salida (IN OUT) debe tener un valor proporcionado fuera del procedimiento, pero puede cambiar dentro de éste. La especificación del tipo de datos no debe incluir ninguna limitación, como longitud. Por ejemplo, para un parámetro de hilera debe usar el tipo de dato VARCHAR2. No tiene que proporcionar la longitud en la especificación del tipo de dato para un parámetro.

#### Procedure Structure:

```
CREATE [OR REPLACE] PROCEDURE ProcedureName
    [ ( Parameter1, . . . , ParameterN ) ]
    IS
        [ sequence of declarations ]
    BEGIN
        sequence of statements
    [ EXCEPTION
        sequence of statements to respond to exceptions ]
    END;
```

Como ejemplo sencillo, el procedimiento *pr\_InsertRegistration* del ejemplo 11.13 inserta una fila en la tabla *Registration* de la base de datos de la universidad. Los parámetros de entrada ofrecen los valores a insertar. El procedimiento *Dbms\_Output.Put\_Line* despliega el mensaje de que la inserción se realizó con éxito. En el código de prueba que sigue a la sentencia CREATE PROCEDURE, la sentencia ROLLBACK elimina el efecto de cualquier sentencia SQL. Las sentencias ROLLBACK son útiles al probar el código cuando los cambios en la base de datos no deben ser permanentes.

#### EJEMPLO 11.13 Procedimiento para insertar una fila en la tabla *Registration* con código para probar el procedimiento

```
CREATE OR REPLACE PROCEDURE pr_InsertRegistration
(aRegNo IN Registration.RegNo%TYPE,
aStdSSN IN Registration.StdSSN%TYPE,
aRegStatus IN Registration.RegStatus%TYPE,
aRegDate IN Registration.RegDate%TYPE,
aRegTerm IN Registration.RegTerm%TYPE,
aRegYear IN Registration.RegYear%TYPE) IS
-- Create a new registration
BEGIN
    INSERT INTO Registration
        (RegNo, StdSSN, RegStatus, RegDate, RegTerm, RegYear)
    VALUES (aRegNo, aStdSSN, aRegStatus, aRegDate, aRegTerm, aRegYear);
```

```

dbms_output.put_line('Added a row to the Registration table');

END;
/

-- Testing code
SET SERVEROUTPUT ON;
-- Number of rows before the procedure execution
SELECT COUNT(*) FROM Registration;

BEGIN
pr_InsertRegistration
(1275,'901-23-4567','F',To_Date('27-Feb-2006'), 'Spring', 2006);
END;
/
-- Number of rows after the procedure execution
SELECT COUNT(*) FROM Registration;
-- Delete the inserted row using the ROLLBACK statement
ROLLBACK;

```

---

Para permitir que otros procedimientos vuelvan a utilizar *pr\_InsertRegistration*, deberá reemplazar la pantalla de salida con un parámetro de salida que indique el éxito o fracaso de la inserción. El ejemplo 11.14 modifica el ejemplo 11.13 para usar un parámetro de salida. La excepción OTHERS detecta gran variedad de errores, como una violación a la limitación de la llave primaria o una limitación de llave externa. Debe usar la excepción OTHERS cuando no necesite un código especializado para cada tipo de excepción. Para detectar un error específico, tiene que utilizar una excepción predefinida (tabla 11.4) o crear una excepción definida por el usuario. En una sección posterior se ejemplifica la excepción definida por el usuario. Después del procedimiento, el *script* incluye casos de prueba para una inserción exitosa, así como una violación a la limitación de la llave primaria.

#### **EJEMPLO 11.14 Procedimiento para insertar una fila en la tabla *Registration* junto con un código para probar el procedimiento**

```

CREATE OR REPLACE PROCEDURE pr_InsertRegistration
(aRegNo IN Registration.RegNo%TYPE,
aStdSSN IN Registration.StdSSN%TYPE,
aRegStatus IN Registration.RegStatus%TYPE,
aRegDate IN Registration.RegDate%TYPE,
aRegTerm IN Registration.RegTerm%TYPE,
aRegYear IN Registration.RegYear%TYPE,
aResult OUT BOOLEAN ) IS
-- Create a new registration
-- aResult is TRUE if successful, false otherwise.
BEGIN
aResult := TRUE;
INSERT INTO Registration
(RegNo, StdSSN, RegStatus, RegDate, RegTerm, RegYear)
VALUES (aRegNo, aStdSSN, aRegStatus, aRegDate, aRegTerm, aRegYear);

```

```

EXCEPTION
WHEN OTHERS THEN aResult := FALSE;
END;
/

-- Testing code
SET SERVEROUTPUT ON;
-- Number of rows before the procedure execution
SELECT COUNT(*) FROM Registration;
DECLARE
    -- Output parameter must be declared in the calling block
    Result BOOLEAN;
BEGIN
    -- This test should succeed.
    -- Procedure assigns value to the output parameter (Result).
    pr_InsertRegistration
    (1275,'901-23-4567','F',To_Date('27-Feb-2006'),'Spring',2006,Result);
    IF Result THEN
        dbms_output.put_line('Added a row to the Registration table');
    ELSE
        dbms_output.put_line('Row not added to the Registration table');
    END IF;

    -- This test should fail because of the duplicate primary key.
    pr_InsertRegistration
    (1275,'901-23-4567','F',To_Date('27-Feb-2006'), 'Spring',2006,Result);
    IF Result THEN
        dbms_output.put_line('Added a row to the Registration table');
    ELSE
        dbms_output.put_line('Row not added to the Registration table');
    END IF;
    END;
/
-- Number of rows after the procedure executions
SELECT COUNT(*) FROM Registration;
-- Delete inserted row
ROLLBACK;

```

**TABLA 11.4**  
**Lista de excepciones comunes predefinidas en PL/SQL**

Excepción	Cuando surge
Cursor_Already_Open	Trata de abrir un cursor abierto anteriormente
Dup_Val_On_Index	Trata de guardar un valor duplicado en un índice único
Invalid_Cursor	Trata de realizar una operación inválida en un cursor, como cerrar un cursor que no se abrió anteriormente
No_Data_Found	La sentencia SELECT INTO no regresa ninguna fila
Rowtype_Mismatch	Trata de asignar valores con tipos de datos incompatibles entre un cursor y una variable
Timeout_On_Resource	Ocurre un tiempo fuera como cuando esperamos un candado exclusivo <sup>1</sup>
Too_Many_Rows	La sentencia SELECT INTO regresa más de una fila

<sup>1</sup> El capítulo 15 explica el uso de tiempos fuera con candados de transacciones para evitar estancamientos.

**procedimientos****versus funciones**

utilice un procedimiento si el código debe tener más de un resultado o un efecto secundario. Las funciones deben poder usarse en expresiones, lo que quiere decir que una llamada de función se puede reemplazar con el valor que regresa. Para que las funciones se puedan usar en expresiones, éstas sólo deben utilizar parámetros de entrada.

## 11.2.2 Funciones PL/SQL

Las funciones deben regresar valores en lugar de manipular las variables de salida y tener efectos secundarios como la inserción de filas en una tabla. Siempre deberá usar un procedimiento si quiere tener más de un resultado y/o tener un efecto secundario. Las funciones se deben poder usar en expresiones, lo que significa que una función debe poder reemplazarse con el valor que regresa. Una función PL/SQL es similar a un procedimiento en el que ambos contienen una lista de parámetros. Sin embargo, una función sólo debe usar parámetros de entrada. Después de la lista de parámetros, el tipo de dato regresado se define sin limitación alguna, como el tamaño. En el cuerpo de la función la secuencia de sentencias debe incluir una sentencia RETURN para generar el valor de salida de la función.

**Estructura de función:**

```
CREATE [OR REPLACE] FUNCTION FunctionName
    [ (Parameter1, . . . , ParameterN) ]
    RETURN DataType
    IS
        [ sequence of declarations ]
    BEGIN
        sequence of statements including a RETURN statement
    [ EXCEPTION
        sequence of statements to respond to exceptions ]
    END;
```

Como ejemplo sencillo, la función *fn\_RetrieveStdName* en el ejemplo 11.15 recupera el nombre de un estudiante proporcionando el número de seguridad social. La excepción predefinida *No\_Data\_Found* es verdadera si la sentencia SELECT no regresa por lo menos una fila. La sentencia SELECT utiliza la cláusula INTO para asociar las variables con las columnas de la base de datos. La cláusula INTO sólo se puede usar cuando la sentencia SELECT regresa cuando mucho una fila. Si se usa una cláusula INTO cuando una sentencia SELECT regresa más de una fila, se genera una excepción. El procedimiento *Raise\_Application\_Error* despliega un mensaje de error y un número de error. Este procedimiento predefinido es útil para manejar errores inesperados.

**EJEMPLO 11.15****Función para recuperar el nombre del estudiante proporcionando su número de seguridad social (SSN)**

```
CREATE OR REPLACE FUNCTION fn_RetrieveStdName
(aStdSSN IN Student.StdSSN%type) RETURN VARCHAR2 IS
-- Retrieves the student name (concatenate first and last name)
-- given a student SSN. If the student does not exist,
-- return null.
aFirstName Student.StdFirstName%TYPE;
aLastName Student.StdLastName%TYPE;

BEGIN
SELECT StdFirstName, StdLastName
INTO aFirstName, aLastName
FROM Student
WHERE StdSSN = aStdSSN;

RETURN(aLastName || ', ' || aFirstName);
```

```

EXCEPTION
-- No_Data_Found aparece si la sentencia SELECT no regresa ningún dato.
WHEN No_Data_Found THEN
    RETURN(NULL);

WHEN OTHERS THEN
    raise_application_error(-20001, 'Database error');

END;
/
-- Testing code
SET SERVEROUTPUT ON;
DECLARE
aStdName VARCHAR2(50);
BEGIN
    -- This call should display a student name.
aStdName := fn_RetrieveStdName('901-23-4567');
IF aStdName IS NULL THEN
    dbms_output.put_line('Student not found');
ELSE
    dbms_output.put_line('Name is ' || aStdName);
END IF;

    -- This call should not display a student name.
aStdName := fn_RetrieveStdName('905-23-4567');
IF aStdName IS NULL THEN
    dbms_output.put_line('Student not found');
ELSE
    dbms_output.put_line('Name is ' || aStdName);
END IF;
END;
/

```

El ejemplo 11.16 muestra una función con una consulta más compleja que la función del ejemplo 11.15. El código de prueba contiene dos casos a probar para un estudiante existente y un estudiante no existente, con una sentencia SELECT que usa la función en el resultado. Un beneficio importante de las funciones es que se pueden usar en expresiones en sentencias SELECT.

#### **EJEMPLO 11.16 Función para calcular el GPA ponderado teniendo el número de seguridad social del estudiante y el año**

```

CREATE OR REPLACE FUNCTION fn_ComputeWeightedGPA
(aStdSSN IN Student.StdSSN%TYPE, aYear IN Offering.OffYear%TYPE)
    RETURN NUMBER IS
-- Computes the weighted GPA given a student SSN and year.
-- Weighted GPA is the sum of units times the grade
-- divided by the total units.
-- If the student does not exist, return null.
WeightedGPA NUMBER;

```

```

BEGIN
SELECT SUM (EnrGrade*CrsUnits) / SUM(CrsUnits)
INTO WeightedGPA
FROM Student, Registration, Enrollment, Offering, Course
WHERE Student.StdSSN = aStdSSN
AND Offering.OffYear = aYear
AND Student.StdSSN = Registration.StdSSN
AND Registration.RegNo = Enrollment.RegNo
AND Enrollment.OfferNo = Offering.OfferNo
AND Offering.CourseNo = Course.CourseNo;

RETURN(WeightedGPA);

EXCEPTION
WHEN No_Data_Found THEN
RETURN(NULL);

WHEN OTHERS THEN
raise_application_error(-20001, 'Database error');

END;
/
-- Testing code
SET SERVEROUTPUT ON;
DECLARE
aGPA DECIMAL(3,2);
BEGIN
-- This call should display a weighted GPA.
aGPA := fn_ComputeWeightedGPA('901-23-4567', 2006);
IF aGPA IS NULL THEN
dbms_output.put_line ('Student or enrollments not found');
ELSE
dbms_output.put_line('Weighted GPA is ' || to_char(aGPA));
END IF;

-- This call should not display a weighted GPA.
aGPA := fn_ComputeWeightedGPA('905-23-4567', 2006);
IF aGPA IS NULL THEN
dbms_output.put_line('Student or enrollments not found');
ELSE
dbms_output.put_line('Weighted GPA is ' || to_char(aGPA));
END IF;
END;
/
-- Use the function in a query
SELECT StdSSN, StdFirstName, StdLastName,
fn_ComputeWeightedGPA(StdSSN, 2006) AS WeightedGPA
FROM Student;

```

### 11.2.3 Uso de cursosres

#### cursor PL/SQL

##### implícito

un cursor que no se declara ni se abre en forma explícita. En vez de ello, una versión especial de la sentencia FOR declara, abre, itera y cierra una sentencia SELECT nombrado localmente. No se puede hacer referencia a un cursor implícito fuera de la sentencia FOR en el que se declara.

Los procedimientos y funciones anteriores son sencillos, ya que comprenden la recuperación de una sola fila. Los procedimientos y funciones más complejos comprenden la iteración a través de varias filas utilizando un cursor. PL/SQL comprende la declaración de cursosres (explícita o implícita), una sentencia FOR especializado para la iteración de cursosres, atributos de cursor para indicar las condiciones de las operaciones de cursor y sentencias para realizar acciones sobre los cursosres explícitos. PL/SQL ofrece soporte para cursosres estáticos en los que la sentencia SQL se conoce en el momento de la compilación, así como cursosres dinámicos en los que la sentencia SQL no se determina sino hasta el momento de la ejecución.

El ejemplo 11.17 ilustra un cursor implícito para regresar el rango de clase de un estudiante en un curso. Los cursosres implícitos no se declaran en la sección DECLARE. En vez de ello, se declaran, abren e iteran dentro de la sentencia FOR. En el ejemplo 11.17, la sentencia FOR se itera en cada renglón de la sentencia SELECT utilizando el cursor implícito *EnrollRec*. La sentencia SELECT clasifica el resultado en orden descendente por calificación. La función sale de la sentencia FOR cuando el valor *StdSSN* coincide con el valor del parámetro. El rango de clase se incrementa sólo cuando la calificación cambia, de modo que dos estudiantes con la misma calificación tienen el mismo rango.

#### EJEMPLO 11.17

##### Uso de un cursor implícito para determinar el rango de clase de un estudiante y un curso determinados

```

CREATE OR REPLACE FUNCTION fn_DetermineRank
(aStdSSN IN Student.StdSSN%TYPE, anOfferNo IN Offering.OfferNo%TYPE)
    RETURN INTEGER IS
    -- Determines the class rank for a given a student SSN and OfferNo.
    -- Uses an implicit cursor.
    -- If the student or offering do not exist, return 0.
    TmpRank INTEGER :=0;
    PrevEnrGrade Enrollment.EnrGrade%TYPE := 9.9;
    FOUND BOOLEAN := FALSE;

    BEGIN
    -- Loop through implicit cursor
    FOR EnrollRec IN
        ( SELECT Student.StdSSN, EnrGrade
          FROM Student, Registration, Enrollment
         WHERE Enrollment.OfferNo = anOfferNo
           AND Student.StdSSN = Registration.StdSSN
           AND Registration.RegNo = Enrollment.RegNo
        ORDER BY EnrGrade DESC ) LOOP
        IF EnrollRec.EnrGrade < PrevEnrGrade THEN
            -- Increment the class rank when the grade changes
            TmpRank := TmpRank + 1;
            PrevEnrGrade := EnrollRec.EnrGrade;
        END IF;
        IF EnrollRec.StdSSN = aStdSSN THEN
            Found := TRUE;
            EXIT;
        END IF;
    END LOOP;

```

```

IF Found THEN
    RETURN(TmpRank);
ELSE
    RETURN(0);
END IF;

EXCEPTION
WHEN OTHERS THEN
    raise_application_error(-20001, 'Database error');

END;
/
-- Testing code
SET SERVEROUTPUT ON;
-- Execute query to see test data
SELECT Student.StdSSN, EnrGrade
FROM Student, Registration, Enrollment
WHERE Enrollment.OfferNo = 5679
    AND Student.StdSSN = Registration.StdSSN
    AND Registration.RegNo = Enrollment.RegNo
ORDER BY EnrGrade DESC;
-- Test script
DECLARE
aRank INTEGER;
BEGIN
-- This call should return a rank of 6.
aRank := fn_DetermineRank('789-01-2345', 5679);
IF aRank > 0 THEN
    dbms_output.put_line('Rank is ' || to_char(aRank));
ELSE
    dbms_output.put_line('Student is not enrolled.');
END IF;

-- This call should return a rank of 0.
aRank := fn_DetermineRank('789-01-2005', 5679);
IF aRank > 0 THEN
    dbms_output.put_line('Rank is ' || to_char(aRank));
ELSE
    dbms_output.put_line('Student is not enrolled.');
END IF;
END;
/

```

**cursor PL/SQL****explícito**

un cursor que se declara con la sentencia CURSOR en la sección DECLARE. Los cursosres explícitos casi siempre se manipulan mediante las sentencias OPEN, CLOSE y FETCH. Es posible hacer referencia a los cursosres explícitos en cualquier lugar dentro de la sección BEGIN.

El ejemplo 11.18 ilustra un procedimiento con un cursor explícito para regresar el rango de clase y la calificación de un estudiante en un curso. El cursor explícito *EnrollCursor* en la sentencia CURSOR contiene el número de curso como parámetro. Los cursosres explícitos deben usar parámetros para los valores de búsqueda no constantes en la sentencia SELECT asociado. Las sentencias OPEN, FETCH y CLOSE reemplazan a la sentencia FOR en el ejemplo 11.17. Después de la sentencia FETCH, la condición *EnrollCursor%NotFound* prueba el cursor vacío.

**EJEMPLO 11.18 Uso de un cursor explícito para determinar el rango de clase y la calificación de un estudiante y curso determinados**

```

CREATE OR REPLACE PROCEDURE pr_DetermineRank
(aStdSSN IN Student.StdSSN%TYPE, anOfferNo IN Offering.OfferNo%TYPE,
 OutRank OUT INTEGER, OutGrade OUT Enrollment.EnrGrade%TYPE ) IS
-- Determines the class rank and grade for a given a student SSN
-- and OfferNo using an explicit cursor.
-- If the student or offering do not exist, return 0.
TmpRank INTEGER :=0;
PrevEnrGrade Enrollment.EnrGrade%TYPE := 9.9;
Found BOOLEAN := FALSE;
TmpGrade Enrollment.EnrGrade%TYPE;
TmpStdSSN Student.StdSSN%TYPE;
-- Explicit cursor
CURSOR EnrollCursor (tmpOfferNo Offering.OfferNo%TYPE) IS
    SELECT Student.StdSSN, EnrGrade
        FROM Student, Registration, Enrollment
       WHERE Enrollment.OfferNo = anOfferNo
         AND Student.StdSSN = Registration.StdSSN
         AND Registration.RegNo = Enrollment.RegNo
      ORDER BY EnrGrade DESC;

BEGIN
-- Open and loop through explicit cursor
OPEN EnrollCursor(anOfferNo);
LOOP
    FETCH EnrollCursor INTO TmpStdSSN, TmpGrade;
    EXIT WHEN EnrollCursor%NotFound;
    IF TmpGrade < PrevEnrGrade THEN
        -- Increment the class rank when the grade changes
        TmpRank := TmpRank + 1;
        PrevEnrGrade := TmpGrade;
    END IF;
    IF TmpStdSSN = aStdSSN THEN
        Found := TRUE;
        EXIT;
    END IF;
END LOOP;

CLOSE EnrollCursor;
IF Found THEN
    OutRank := TmpRank;
    OutGrade := PrevEnrGrade;
ELSE
    OutRank := 0;
    OutGrade := 0;
END IF;

```

```

EXCEPTION
WHEN OTHERS THEN
    raise_application_error(-20001, 'Database error');
END;
/
-- Testing code
SET SERVEROUTPUT ON;
-- Execute query to see test data
SELECT Student.StdSSN, EnrGrade
    FROM Student, Registration, Enrollment
   WHERE Student.StdSSN = Registration.StdSSN
     AND Registration.RegNo = Enrollment.RegNo
     AND Enrollment.OfferNo = 5679
ORDER BY EnrGrade DESC;

-- Test script
DECLARE
aRank INTEGER;
aGrade Enrollment.EnrGrade%TYPE;
BEGIN
-- This call should produce a rank of 6.
pr_DetermineRank('789-01-2345', 5679, aRank, aGrade);
IF aRank > 0 THEN
    dbms_output.put_line('Rank is ' || to_char(aRank) || '.');
    dbms_output.put_line('Grade is ' || to_char(aGrade) || '.');
ELSE
    dbms_output.put_line('Student is not enrolled.');
END IF;
-- This call should produce a rank of 0.
pr_DetermineRank('789-01-2005', 5679, aRank, aGrade);
IF aRank > 0 THEN
    dbms_output.put_line('Rank is ' || to_char(aRank) || '.');
    dbms_output.put_line('Grade is ' || to_char(aGrade) || '.');
ELSE
    dbms_output.put_line('Student is not enrolled.');
END IF;
END;
/

```

PL/SQL ofrece soporte para varios atributos de cursor, como ilustra la tabla 11.5. Cuando se utiliza con un cursor explícito, el nombre de cursor va antes del atributo. Cuando se usa con un cursor implícito, la palabra clave SQL aparece antes del atributo del cursor. Por ejemplo, *SQL%RowCount* indica el número de filas en un cursor implícito. No se usa el nombre del cursor implícito.

#### 11.2.4 Paquetes PL/SQL

Los paquetes soportan una unidad de módulos más extensa que los procedimientos o funciones. Un paquete puede contener procedimientos, funciones, excepciones, variables, constantes, tipos y cursores. Al agrupar objetos relacionados, un paquete facilita el reuso más que los

**TABLA 11.5**  
Lista de atributos comunes de cursor

Atributo de cursor	Valor
%IsOpen	Verdadero si el cursor está abierto
%Found	Verdadero si el cursor no está vacío después de un sentencia FETCH
%NotFound	Verdadero si el cursor está vacío después de una sentencia FETCH
%RowCount	Número de filas buscadas. Después de cada FETCH, RowCount se incrementa

procedimientos y funciones de manera individual. Oracle ofrece muchos paquetes predefinidos, como el paquete *DBMS\_Output*, que contiene grupos de objetos relacionados. Además, un paquete separa una interfaz pública de una implementación privada para soportar menores esfuerzos de mantenimiento de software. Los cambios a una implementación privada no afectan el uso de un paquete a través de su interfaz. El capítulo 18 sobre objetos de bases de datos proporciona más detalles sobre los beneficios de unidades modulares más extensas.

La interfaz de un paquete contiene las definiciones de los procedimientos y funciones, además de otros objetos que es posible especificar en la sección DECLARE de un bloque PL/SQL. Todos los objetos en la interfaz de un paquete son públicos. El ejemplo 11.19 demuestra la interfaz para un paquete que combina algunos de los procedimientos y funciones presentadas en secciones anteriores.

**Package Interface Structure:**

```
CREATE [OR REPLACE] PACKAGE PackageName IS
  [ Constant, variable, and type declarations ]
  [ Cursor declarations ]
  [ Exception declarations ]
  [ Procedure definitions ]
  [ Function definitions ]
END PackageName;
```

**EJEMPLO 11.19 Interfaz de un paquete que contiene procedimientos y funciones relacionadas para la base de datos de la universidad**

```
CREATE OR REPLACE PACKAGE pck_University IS
  PROCEDURE pr_DetermineRank
    (aStdSSN IN Student.StdSSN%TYPE, anOfferNo IN Offering.OfferNo%TYPE,
     OutRank OUT INTEGER, OutGrade OUT Enrollment.EnrGrade%TYPE );
  FUNCTION fn_ComputeWeightedGPA
    (aStdSSN IN Student.StdSSN%TYPE, aYear IN Offering.OffYear%TYPE)
      RETURN NUMBER;
END pck_University;
/
```

La implementación o el cuerpo de un paquete contienen sus detalles privados. Para cada objeto en la interfaz empaquetada, el cuerpo del paquete debe definir una implementación. Además, los objetos privados pueden definirse en el cuerpo del paquete. Los objetos privados pueden usarse sólo dentro del cuerpo del paquete. Los usuarios externos de un paquete no tienen acceso a los objetos privados. El ejemplo 11.20 muestra el cuerpo para la interfaz de paquete del ejemplo 11.19. Cabe señalar que cada procedimiento o función termina con una sentencia END que contiene el nombre del procedimiento o función. Por lo demás, el procedimiento y la implementación de la función son idénticas a la creación de un procedimiento o función fuera de un paquete.

**Estructura del cuerpo de un paquete:**

```

CREATE [OR REPLACE] PACKAGE BODY PackageName IS
[ Variable and type declarations ]
[ Cursor declarations ]
[ Exception declarations ]
[ Procedure implementations ]
[ Function implementations ]
[ BEGIN sequence of statements ]
[ EXCEPTION exception handling statements ]
END PackageName;

```

**EJEMPLO 11.20****Cuerpo de un paquete que contiene implementaciones de procedimientos y funciones**

```

CREATE OR REPLACE PACKAGE BODY pck_University IS
PROCEDURE pr_DetermineRank
(aStdSSN IN Student.StdSSN%TYPE, anOfferNo IN Offering.OfferNo%TYPE,
OutRank OUT INTEGER, OutGrade OUT Enrollment.EnrGrade%TYPE ) IS
-- Determines the class rank and grade for a given a student SSN
-- and OfferNo using an explicit cursor.
-- If the student or offering do not exist, return 0.
TmpRank INTEGER :=0;
PrevEnrGrade Enrollment.EnrGrade%TYPE := 9.9;
Found BOOLEAN := FALSE;
TmpGrade Enrollment.EnrGrade%TYPE;
TmpStdSSN Student.StdSSN%TYPE;
-- Explicit cursor
CURSOR EnrollCursor (tmpOfferNo Offering.OfferNo%TYPE) IS
SELECT Student.StdSSN, EnrGrade
FROM Student, Registration, Enrollment
WHERE Enrollment.OfferNo = anOfferNo
AND Student.StdSSN = Registration.StdSSN
AND Registration.RegNo = Enrollment.RegNo
ORDER BY EnrGrade DESC;

BEGIN
-- Open and loop through explicit cursor
OPEN EnrollCursor(anOfferNo);
LOOP
FETCH EnrollCursor INTO TmpStdSSN, TmpGrade;
EXIT WHEN EnrollCursor%NotFound;
IF TmpGrade < PrevEnrGrade THEN
-- Increment the class rank when the grade changes
TmpRank := TmpRank + 1;
PrevEnrGrade := TmpGrade;
END IF;
IF TmpStdSSN = aStdSSN THEN
Found := TRUE;
EXIT;
END IF;
END LOOP;

```

```

CLOSE EnrollCursor;
IF Found THEN
    OutRank := TmpRank;
    OutGrade := PrevEnrGrade;
ELSE
    OutRank := 0;
    OutGrade := 0;
END IF;

EXCEPTION
WHEN OTHERS THEN
    raise_application_error(-20001, 'Database error');
END pr_DetermineRank;

FUNCTION fn_ComputeWeightedGPA
(aStdSSN IN Student.StdSSN%TYPE, aYear IN Offering.OffYear%TYPE)
    RETURN NUMBER IS
-- Computes the weighted GPA given a student SSN and year.
-- Weighted GPA is the sum of units times the grade
-- divided by the total units.
-- If the student does not exist, return null.
    WeightedGPA NUMBER;

BEGIN

SELECT SUM(EnrGrade*CrsUnits)/SUM(CrsUnits)
INTO WeightedGPA
FROM Student, Registration, Enrollment, Offering, Course
WHERE Student.StdSSN = aStdSSN
    AND Offering.OffYear = aYear
    AND Student.StdSSN = Registration.StdSSN
    AND Registration.RegNo = Enrollment.RegNo
    AND Enrollment.OfferNo = Offering.OfferNo
    AND Offering.CourseNo = Course.CourseNo;

RETURN(WeightedGPA);

EXCEPTION
WHEN no_data_found THEN
    RETURN(NULL);
WHEN OTHERS THEN
    raise_application_error(-20001, 'Database error');
END fn_ComputeWeightedGPA;
END pck_University;
/

```

Para utilizar los objetos en un paquete, necesita usar el nombre del paquete antes del nombre del objeto. Observe que en el ejemplo 11.21, el nombre del paquete (*pck\_University*) va antes del nombre del procedimiento y de la función.

**EJEMPLO 11.21 Guión para utilizar los procedimientos y funciones del paquete universitario**

```

SET SERVEROUTPUT ON;
DECLARE
aRank INTEGER;
aGrade Enrollment.EnrGrade%TYPE;
aGPA NUMBER;
BEGIN
-- This call should produce a rank of 6.
pck_University.pr_DetermineRank('789-01-2345', 5679, aRank, aGrade);
IF aRank > 0 THEN
    dbms_output.put_line('Rank is ' || to_char(aRank) || '.');
    dbms_output.put_line('Grade is ' || to_char(aGrade) || '.');
ELSE
    dbms_output.put_line('Student is not enrolled.');
END IF;
-- This call should display a weighted GPA.
aGPA := pck_University.fn_ComputeWeightedGPA('901-23-4567', 2006);
IF aGPA IS NULL THEN
    dbms_output.put_line('Student or enrollments not found');
ELSE
    dbms_output.put_line('Weighted GPA is ' || to_char(aGPA));
END IF;
END;
/

```

## 11.3 Disparadores (*triggers*)

**disparador (trigger)** una regla que un DBMS guarda y ejecuta. Como un disparador implica un evento, una condición y una secuencia de acciones, también se conoce como regla de evento-condición-acción.

Los **disparadores** son reglas que maneja un DBMS. Como un disparador implica un evento, una condición y una secuencia de acciones, también se conoce como regla de evento-condición-acción. La escritura de la parte de la acción o el cuerpo del disparador es similar a la de un procedimiento o función, sólo que un disparador no tiene parámetros. Los disparadores se ejecutan mediante el sistema de reglas del DBMS y no a través de llamadas explícitas, como en el caso de los procedimientos y las funciones. Los disparadores se volvieron parte oficial del SQL:1999, aunque la mayor parte de los distribuidores de DBMS los implementaron mucho antes del lanzamiento de esa versión del SQL.

Esta sección abarca los disparadores de Oracle con antecedentes en los disparadores de SQL:2003. La primera parte analiza las razones por las que los disparadores son elementos importantes del desarrollo de aplicaciones de bases de datos y proporciona su clasificación. La segunda parte muestra su codificación en PL/SQL, y la última presenta los procedimientos de ejecución de disparadores de Oracle y SQL:2003.

### 11.3.1 Motivación y clasificación de los disparadores

Los disparadores se implementan con frecuencia en los DBMS porque tienen gran variedad de usos en las aplicaciones empresariales. La siguiente lista explica sus usos típicos.

- **Restricciones de integridad compleja:** Restricciones de integridad que no se pueden especificar mediante restricciones en las sentencias CREATE TABLE. Una restricción típica entre las limitaciones de las sentencias CREATE TABLE es que no se pueden referenciar las columnas de otras tablas. Los disparadores permiten hacer referencia a columnas de varias tablas para superar esta limitación. Una alternativa para un disparador con una limitación compleja es una aserción, que se analizan en el capítulo 14. Sin embargo, la mayor

parte de los DBMS no ofrecen soporte para las aserciones, de modo que los disparadores constituyen la única opción para las restricciones de integridad compleja.

- **Restricciones de transición:** Restricciones de integridad que comparan los valores antes y después de una actualización. Por ejemplo, puede escribir un disparador para implementar la limitación de transición para que los aumentos de salario no superen el 10 por ciento.
- **Propagación de actualizaciones:** Columnas derivadas de una actualización en tablas relacionadas, como el hecho de mantener un saldo perpetuo en el inventario o los asientos que quedan vacíos en la programación de un vuelo.
- **Reporte de excepciones:** Crear un registro de condiciones poco comunes como alternativa para rechazar una transacción. Un disparador también puede enviar una notificación en un mensaje de correo electrónico. Por ejemplo, en lugar de rechazar un incremento salarial de 10 por ciento, un disparador puede crear un registro de excepción e informar al gerente que revise ese aumento.
- **Seguimiento de auditorías:** Crear un registro histórico de una transacción como el histórico de uso de un cajero automático.

SQL:2003 clasifica los disparadores por granularidad, momento y evento aplicable. Según su granularidad, un disparador puede comprender cada una de las filas afectadas por una sentencia SQL o una sentencia SQL completo. Los disparadores de filas son más comunes que los de sentencia. De acuerdo con el momento, un disparador se puede accionar antes o después de un evento. Por lo regular, los disparadores para revisar las restricciones se disparan antes de un evento, mientras que aquellos que actualizan tablas relacionadas y otras acciones se disparan después de un evento. Para un evento aplicable, un disparador se puede aplicar a las sentencias INSERT, UPDATE y DELETE. Los disparadores de actualización pueden especificar una lista de columnas aplicables.

Como la especificación de disparadores de SQL:1999 se definió en respuesta a las implementaciones de los distribuidores, casi todas las implementaciones de disparadores varían de la especificación original en SQL:1999 y la especificación revisada en SQL:2003. Oracle ofrece soporte para casi todas las partes de la especificación, al tiempo que agrega extensiones propietarias. Una extensión importante es el disparador INSTEAD OF que se activa en lugar de un evento, no antes ni después de éste. Oracle también ofrece soporte para eventos de definición de datos y otros eventos de base de datos. Microsoft SQL Server proporciona disparadores de sentencias con acceso a los datos de las filas, en lugar de los disparadores de fila. Por tanto, la mayoría de los DBMS ofrecen soporte para la especificación de disparadores de SQL:2003 con base en la granularidad, el momento y los eventos aplicables, pero no se apegan en forma estricta a la sintaxis de los disparadores de SQL:2003.

### 11.3.2 Disparadores de Oracle

Un disparador de Oracle contiene un nombre de disparador, una especificación del momento, una cláusula de referencia opcional, granularidad opcional, una cláusula WHEN opcional y un bloque PL/SQL para el cuerpo, como explicamos en la lista siguiente:

- La especificación de momento comprende las palabras clave BEFORE, AFTER o INSTEAD OF, además de un evento disparador que utiliza las palabras clave INSERT, UPDATE o DELETE. Con el evento UPDATE, puede especificar una lista opcional de columnas. Para especificar varios eventos, puede usar la palabra clave OR. Oracle también ofrece soporte para definición de datos y otros eventos de base de datos, pero éstos se encuentran más allá del alcance de este capítulo.
- La cláusula de referencia permite alias para los datos nuevos y antiguos que pueden referenciarse en un disparador.
- La granularidad se especifica mediante las palabras clave FOR EACH ROW. Si omite estas palabras, el disparador es un disparador de sentencia.
- La cláusula WHEN limita el momento en que un disparador se acciona o ejecuta. Como Oracle tiene muchas restricciones sobre las condiciones en las cláusulas WHEN, ésta no se usa con frecuencia.

- El cuerpo de un disparador parece otro bloque PL/SQL, sólo que los disparadores tienen más restricciones para las sentencias en un bloque.

#### **Oracle Trigger Structure:**

```
CREATE [OR REPLACE] TRIGGER TriggerName
TriggerTiming TriggerEvent
[ Referencing clause ]
[ FOR EACH ROW ]
[ WHEN ( Condition ) ]
[ DECLARE sequence of declarative statements ]
BEGIN sequence of statements
[ EXCEPTION exception handling statements ]
END;
```

#### *Disparadores de introducción y código de prueba*

Para empezar con algunos sencillos disparadores de Oracle, los ejemplos 11.22 a 11.24 contienen disparadores que se accionan en cada sentencia INSERT, UPDATE y DELETE, respectivamente, en la tabla *Course*. El ejemplo 11.25 muestra un disparador con un evento combinado que se acciona para cada acción en la tabla *Course*. Los disparadores en los ejemplos 11.22 a 11.25 no tienen ningún propósito además de ilustrar una amplia variedad de sintaxis de disparadores, como explicamos en la siguiente lista.

- Un esquema de nombre común para los disparadores identifica el nombre de la tabla, las acciones de disparo (I para INSERT, U para UPDATE y D para DELETE) y el momento (B para BEFORE y A para AFTER). Por ejemplo, la última parte del nombre del disparador (DIUA) en el ejemplo 11.25 indica los eventos DELETE, INSERT y UPDATE, además del momento AFTER.
- En el ejemplo 11.25, la palabra clave OR en la especificación del evento del disparador soporta eventos compuestos que suponen más de un evento.
- No existe una cláusula de referencia ya que los nombres predeterminados para la fila antigua (*:OLD*) y nueva (*:NEW*) se utilizan en el cuerpo de los disparadores.

#### **EJEMPLO 11.22 Disparador que se lanza para la sentencia INSERT en la tabla Course con el código de prueba para lanzar el disparador**

```
CREATE OR REPLACE TRIGGER tr_Course_IA
AFTER INSERT
ON Course
FOR EACH ROW
BEGIN
    -- No references to OLD row because only NEW exists for INSERT
    dbms_output.put_line('Inserted Row');
    dbms_output.put_line('CourseNo: ' || :NEW.CourseNo);
    dbms_output.put_line('Course Description: ' || :NEW.CrsDesc);
    dbms_output.put_line('Course Units: ' || To_Char(:NEW.CrsUnits));
END;
/
-- Testing statements
SET SERVEROUTPUT ON;
INSERT INTO Course (CourseNo, CrsDesc, CrsUnits)
VALUES ('IS485','Advanced Database Management',4);

ROLLBACK;
```

**EJEMPLO 11.23 Disparador que se lanza para cada sentencia UPDATE en la tabla Course con el código de prueba para lanzar el disparador**

```

CREATE OR REPLACE TRIGGER tr_Course_UA
AFTER UPDATE
ON Course
FOR EACH ROW
BEGIN
    dbms_output.put_line('New Row Values');
    dbms_output.put_line('CourseNo: ' || :NEW.CourseNo);
    dbms_output.put_line('Course Description: ' || :NEW.CrsDesc);
    dbms_output.put_line('Course Units: ' || To_Char(:NEW.CrsUnits));

    dbms_output.put_line('Old Row Values');
    dbms_output.put_line('CourseNo: ' || :OLD.CourseNo);
    dbms_output.put_line('Course Description: ' || :OLD.CrsDesc);
    dbms_output.put_line('Course Units: ' || To_Char(:OLD.CrsUnits));

END;
/
-- Testing statements
SET SERVEROUTPUT ON;
-- Add row so it can be updated
INSERT INTO Course (CourseNo, CrsDesc, CrsUnits)
VALUES ('IS485','Advanced Database Management',4);

UPDATE Course
SET CrsUnits = 3
WHERE CourseNo = 'IS485';

ROLLBACK;

```

**EJEMPLO 11.24 Disparador que se lanza para cada sentencia DELETE en la tabla Course con el código de prueba para lanzar el disparador**

```

CREATE OR REPLACE TRIGGER tr_Course_DA
AFTER DELETE
ON Course
FOR EACH ROW
BEGIN
    -- No references to NEW row because only OLD exists for DELETE
    dbms_output.put_line('Deleted Row');
    dbms_output.put_line('CourseNo: ' || :OLD.CourseNo);
    dbms_output.put_line('Course Description: ' || :OLD.CrsDesc);
    dbms_output.put_line('Course Units: ' || To_Char(:OLD.CrsUnits));

END;
/
-- Testing statements
SET SERVEROUTPUT ON;
-- Insert row so that it can be deleted
INSERT INTO Course (CourseNo, CrsDesc, CrsUnits)
-VALUES ('IS485','Advanced Database Management',4);

```

```
DELETE FROM Course
WHERE CourseNo = 'IS485';

ROLLBACK;
```

**EJEMPLO 11.25 Disparador con un evento combinado que se lanza para cada acción en la tabla Course con el código de prueba para lanzar el disparador**

```
CREATE OR REPLACE TRIGGER tr_Course_DIUA
AFTER INSERT OR UPDATE OR DELETE
ON Course
FOR EACH ROW
BEGIN
    dbms_output.put_line('Inserted Table');
    dbms_output.put_line('CourseNo: ' || :NEW.CourseNo);
    dbms_output.put_line('Course Description: ' || :NEW.CrsDesc);
    dbms_output.put_line('Course Units: ' || To_Char(:NEW.CrsUnits));

    dbms_output.put_line('Deleted Table');
    dbms_output.put_line('CourseNo: ' || :OLD.CourseNo);
    dbms_output.put_line('Course Description: ' || :OLD.CrsDesc);
    dbms_output.put_line('Course Units: ' || To_Char(:OLD.CrsUnits));
END;
/
-- Testing statements
SET SERVEROUTPUT ON;
INSERT INTO Course (CourseNo, CrsDesc, CrsUnits)
VALUES ('IS485','Advanced Database Management',4);

UPDATE Course
SET CrsUnits = 3
WHERE CourseNo = 'IS485';

DELETE FROM Course
WHERE CourseNo = 'IS485';

ROLLBACK;
```

A diferencia de los procedimientos, los disparadores no se pueden probar directamente. En vez de ello, utilice sentencias SQL que hagan que se lance el disparador. Cuando el disparador del ejemplo 11.25 se lanza para una sentencia INSERT, los valores antiguos quedan nulos. De modo similar, cuando el disparador se lanza para una sentencia DELETE, los nuevos valores quedan nulos. Cuando el disparador se lanza para una sentencia UPDATE, los valores antiguos y nuevos son nulos, a menos que la tabla haya tenido valores nulos antes de la actualización.

*Disparador BEFORE ROW para revisar las restricciones*

Por lo regular, los disparadores BEFORE ROW se usan para restricciones de integridad compleja porque no deben contener sentencias de manipulación SQL. Por ejemplo, la inscripción en un curso comprende una limitación de integridad compleja para asegurar que exista un lugar en el curso relacionado. El ejemplo 11.26 muestra un disparador BEFORE ROW para asegurar que todavía hay un lugar en el momento en que un estudiante se inscribe en un curso. El disparador se asegura de que el número de estudiantes inscritos en el curso es menor que el límite. El código

de prueba inserta a los estudiantes y modifica el número de estudiantes inscritos, de modo que la próxima inserción dé lugar a un error. El disparador utiliza una excepción definida por el usuario para manejar el error.

**EJEMPLO 11.26 Disparador para asegurarse de que queda un lugar en un curso**

```

CREATE OR REPLACE TRIGGER tr_Enrollment_IB
-- This trigger ensures that the number of enrolled
-- students is less than the offering limit.
BEFORE INSERT
ON Enrollment
FOR EACH ROW
DECLARE
    anOffLimit Offering.OffLimit%TYPE;
    anOffNumEnrolled Offering.OffNumEnrolled%TYPE;
    -- user defined exception declaration
    NoSeats EXCEPTION;
    ExMessage VARCHAR(200);
BEGIN
    SELECT OffLimit, OffNumEnrolled
        INTO anOffLimit, anOffNumEnrolled
        FROM Offering
        WHERE Offering.OfferNo = :NEW.OfferNo;

    IF anOffNumEnrolled >= anOffLimit THEN
        RAISE NoSeats;
    END IF;
EXCEPTION
    WHEN NoSeats THEN
        -- error number between -20000 and -20999
        ExMessage := 'No seats remaining in offering ' ||
                     to_char(:NEW.OfferNo) || '.';
        ExMessage := ExMessage || 'Number enrolled: ' ||
                     to_char(anOffNumEnrolled) || '.';
        ExMessage := ExMessage || 'Offering limit: ' ||
                     to_char(anOffLimit);
        Raise_Application_Error(-20001, ExMessage);
    END;
/
-- Testing statements
SET SERVEROUTPUT ON;
-- See offering limit and number enrolled
SELECT OffLimit, OffNumEnrolled
    FROM Offering
    WHERE Offering.OfferNo = 5679;
-- Insert the last student
INSERT INTO Enrollment (RegNo, OfferNo, EnrGrade)
    VALUES (1234,5679,0);

-- update the number of enrolled students
UPDATE Offering
    SET OffNumEnrolled = OffNumEnrolled + 1
    WHERE OfferNo = 5679;

```

```
-- See offering limit and number enrolled
SELECT OffLimit, OffNumEnrolled
  FROM Offering
 WHERE Offering.OfferNo = 5679;
-- Insert a student beyond the limit
INSERT INTO Enrollment (RegNo, OfferNo, EnrGrade)
  VALUES (1236,5679,0);

ROLLBACK;
```

*Disparador AFTER ROW para la propagación de actualizaciones*

El código de prueba para el disparador BEFORE ROW en el ejemplo 11.26 incluye una sentencia UPDATE para incrementar el número de estudiantes inscritos. Un disparador AFTER puede automatizar esta tarea, como muestra el ejemplo 11.27. Los disparadores en los ejemplos 11.26

---

**EJEMPLO 11.27 Disparador para actualizar el número de estudiantes inscritos en un curso**

```
CREATE OR REPLACE TRIGGER tr_Enrollment_IA
-- This trigger updates the number of enrolled
-- students the related offering row.
AFTER INSERT
  ON Enrollment
  FOR EACH ROW
BEGIN
    UPDATE Offering
      SET OffNumEnrolled = OffNumEnrolled + 1
        WHERE OfferNo = :NEW.OfferNo;
EXCEPTION
  WHEN OTHERS THEN
    RAISE_Application_Error(-20001, 'Database error');
END;
/
-- Testing statements
SET SERVEROUTPUT ON;
-- See the offering limit and number enrolled
SELECT OffLimit, OffNumEnrolled
  FROM Offering
 WHERE Offering.OfferNo = 5679;
-- Insert the last student
INSERT INTO Enrollment (RegNo, OfferNo, EnrGrade)
  VALUES (1234,5679,0);

-- See the offering limit and number enrolled
SELECT OffLimit, OffNumEnrolled
  FROM Offering
 WHERE Offering.OfferNo = 5679;

ROLLBACK;
```

y 11.27 trabajan en combinación. El disparador BEFORE ROW se asegura de que todavía haya un lugar en el curso. El disparador AFTER ROW actualiza la fila *Offering* relacionada.

#### *Combinando eventos de disparador para reducir el número de disparadores*

Los disparadores en los ejemplos 11.26 y 11.27 comprenden inserciones en la tabla *Enrollment*. Se necesitan disparadores adicionales para las actualizaciones de la columna *Enrollment*, *OfferNo* y las eliminaciones de las filas *Enrollment*.

Como alternativa para separar los disparadores para los eventos en la misma tabla, es posible escribir disparadores extensos BEFORE y AFTER. Cada disparador contiene varios eventos, como muestran los ejemplos 11.28 y 11.29. La parte de la acción del disparador en el ejemplo 11.29 utiliza las palabras clave INSERTING, UPDATING y DELETING para determinar el evento disparador. El script en el ejemplo 11.30 es complejo porque prueba dos disparadores complejos.

#### EJEMPLO 11.28

#### **Disparador para asegurarse de que queda un lugar en un curso al insertar o actualizar una fila Enrollment**

```
-- Drop the previous trigger to avoid interactions
DROP TRIGGER tr_Enrollment_IB;
CREATE OR REPLACE TRIGGER tr_Enrollment_IUB
-- This trigger ensures that the number of enrolled
-- students is less than the offering limit.
BEFORE INSERT OR UPDATE OF OfferNo
ON Enrollment
FOR EACH ROW
DECLARE
    anOffLimit Offering.OffLimit%TYPE;
    anOffNumEnrolled Offering.OffNumEnrolled%TYPE;
    NoSeats EXCEPTION;
    ExMessage VARCHAR(200);
BEGIN
    SELECT OffLimit, OffNumEnrolled
        INTO anOffLimit, anOffNumEnrolled
        FROM Offering
        WHERE Offering.OfferNo = :NEW.OfferNo;

    IF anOffNumEnrolled >= anOffLimit THEN
        RAISE NoSeats;
    END IF;
EXCEPTION
    WHEN NoSeats THEN
        -- error number between -20000 and -20999
        ExMessage := 'No seats remaining in offering ' ||
                    to_char(:NEW.OfferNo) || ':';
        ExMessage := ExMessage || 'Number enrolled: ' ||
                    to_char(anOffNumEnrolled) || '. ';
        ExMessage := ExMessage || 'Offering limit: ' ||
                    to_char(anOffLimit);
        raise_application_error(-20001, ExMessage);
END;
```

**EJEMPLO 11.29 Disparador para actualizar el número de estudiantes inscritos en un curso cuando se inserta, actualiza o elimina una fila Enrollment**

```
-- Drop the previous trigger to avoid interactions
DROP TRIGGER tr_Enrollment_IA;
CREATE OR REPLACE TRIGGER tr_Enrollment_DIU
-- This trigger updates the number of enrolled
-- students the related offering row.
AFTER INSERT OR DELETE OR UPDATE of OfferNo
ON Enrollment
FOR EACH ROW
BEGIN
    -- Increment the number of enrolled students for insert, update
    IF INSERTING OR UPDATING THEN
        UPDATE Offering
            SET OffNumEnrolled = OffNumEnrolled + 1
            WHERE OfferNo = :NEW.OfferNo;
    END IF;
    -- Decrease the number of enrolled students for delete, update
    IF UPDATING OR DELETING THEN
        UPDATE Offering
            SET OffNumEnrolled = OffNumEnrolled - 1
            WHERE OfferNo = :OLD.OfferNo;
    END IF;

    EXCEPTION
        WHEN OTHERS THEN
            raise_application_error(-20001, 'Database error');
END;
```

**EJEMPLO 11.30 Script para probar los disparadores en los ejemplos 11.28 y 11.29**

```
-- Test case 1
-- See the offering limit and number enrolled
SELECT OffLimit, OffNumEnrolled
    FROM Offering
    WHERE Offering.OfferNo = 5679;
-- Insert the last student
INSERT INTO Enrollment (RegNo, OfferNo, EnrGrade)
    VALUES (1234,5679,0);
-- See the offering limit and the number enrolled
SELECT OffLimit, OffNumEnrolled
    FROM Offering
    WHERE Offering.OfferNo = 5679;

-- Test case 2
-- Insert a student beyond the limit: exception raised
INSERT INTO Enrollment (RegNo, OfferNo, EnrGrade)
    VALUES (1236,5679,0);
-- Transfer a student to offer 5679: exception raised
UPDATE Enrollment
```

```

SET OfferNo = 5679
WHERE RegNo = 1234 AND OfferNo = 1234;

-- Test case 3
-- See the offering limit and the number enrolled before update
SELECT OffLimit, OffNumEnrolled
FROM Offering
WHERE Offering.OfferNo = 4444;
-- Update a student to a non full offering
UPDATE Enrollment
SET OfferNo = 4444
WHERE RegNo = 1234 AND OfferNo = 1234;
-- See the offering limit and the number enrolled after update
SELECT OffLimit, OffNumEnrolled
FROM Offering
WHERE Offering.OfferNo = 4444;

-- Test case 4
-- See the offering limit and the number enrolled before delete
SELECT OffLimit, OffNumEnrolled
FROM Offering
WHERE Offering.OfferNo = 1234;
-- Delete an enrollment
DELETE Enrollment
WHERE OfferNo = 1234;
-- See the offering limit and the number enrolled
SELECT OffLimit, OffNumEnrolled
FROM Offering
WHERE Offering.OfferNo = 1234;

-- Erase all changes
ROLLBACK;

```

No existe una preferencia clara entre muchos disparadores pequeños o pocos disparadores grandes. Aunque los pequeños son más fáciles de entender que los grandes, el número de disparadores es un factor que hace más complicado entender las interacciones entre ellos. La siguiente subsección explica los procedimientos de ejecución de los disparadores para aclarar los aspectos de interacción entre éstos.

#### *Ejemplos adicionales del disparador BEFORE ROW*

Los disparadores BEFORE también se pueden usar para restricciones de transición y estandarización de datos. El ejemplo 11.31 ilustra un disparador para una limitación de transición. El disparador contiene una cláusula WHEN para limitar la ejecución del disparador. El ejemplo 11.32 muestra un disparador para implementar el uso de mayúsculas en las columnas con los nombres de los profesores. Aunque los disparadores BEFORE no deben realizar actualizaciones con sentencias SQL, pueden cambiar los valores nuevos, como lo demuestra el disparador en el ejemplo 11.32.

#### **EJEMPLO 11.31**

#### **Disparador para asegurar que los incrementos salariales no excedan el 10 por ciento**

Debemos hacer notar que las palabras clave NEW y OLD no deben estar precedidas por dos puntos (:) cuando se utilizan en una condición WHEN.

```

CREATE OR REPLACE TRIGGER tr_FacultySalary_UB
-- This trigger ensures that a salary increase does not exceed
-- 10%.
BEFORE UPDATE OF FacSalary
ON Faculty
FOR EACH ROW
WHEN (NEW.FacSalary > 1.1 * OLD.FacSalary)
DECLARE
    SalaryIncreaseTooHigh EXCEPTION;
    ExMessage VARCHAR(200);
BEGIN
    RAISE SalaryIncreaseTooHigh;
EXCEPTION
    WHEN SalaryIncreaseTooHigh THEN
        -- error number between -20000 and -20999
        ExMessage := 'Salary increase exceeds 10%. ';
        ExMessage := ExMessage || 'Current salary: ' ||
                     to_char(:OLD.FacSalary) || '.';
        ExMessage := ExMessage || 'New salary: ' ||
                     to_char(:NEW.FacSalary) || '.';
        Raise_Application_Error(-20001, ExMessage);
END;
/
SET SERVEROUTPUT ON;
-- Test case 1: salary increase of 5%
UPDATE Faculty
    SET FacSalary = FacSalary * 1.05
    WHERE FacSSN = '543-21-0987';
SELECT FacSalary FROM Faculty WHERE FacSSN = '543-21-0987';
-- Test case 2: salary increase of 20% should generate an error.
UPDATE Faculty
    SET FacSalary = FacSalary * 1.20
    WHERE FacSSN = '543-21-0987';
ROLLBACK;

```

### EJEMPLO 11.32 Disparador para cambiar la mayúscula del nombre y apellido de los profesores

```

CREATE OR REPLACE TRIGGER tr_FacultyName_IUB
-- This trigger changes the case of FacFirstName and FacLastName.
BEFORE INSERT OR UPDATE OF FacFirstName, FacLastName
ON Faculty
FOR EACH ROW
BEGIN
    :NEW.FacFirstName := Upper(:NEW.FacFirstName);
    :NEW.FacLastName := Upper(:NEW.FacLastName);
END;
/
-- Testing statements
UPDATE Faculty

```

```

SET FacFirstName = 'Joe', FacLastName = 'Smith'
WHERE FacSSN = '543-21-0987';
-- Display the changed faculty name.
SELECT FacFirstName, FacLastName
FROM Faculty
WHERE FacSSN = '543-21-0987';
ROLLBACK;

```

*Disparador AFTER ROW para reporte de excepciones*

El disparador del ejemplo 11.31 implementa una limitación dura en el hecho de que rechaza los aumentos grandes (superiores a 10 por ciento). Un enfoque más flexible es una limitación suave en la que un aumento grande provoca la escritura de un renglón en una tabla de excepciones. La actualización se lleva a cabo con éxito, pero un administrador puede revisar posteriormente la tabla de excepciones para emprender una acción adicional. También se puede enviar un mensaje para avisar al administrador que debe revisar renglones específicos en la tabla de excepciones. El ejemplo 11.33 ilustra un disparador para implementar una limitación suave para aumentos grandes a los empleados. Se utiliza el momento del disparador AFTER porque una fila sólo debe escribirse en la tabla de excepciones si la actualización tiene éxito. Como demostramos en la siguiente sección, los disparadores AFTER ROW sólo se ejecutan si no se encuentran errores al revisar la limitación de integridad.

**EJEMPLO 11.33 Sentencia CREATE TABLE para la tabla de excepciones y disparador para insertar una fila en la tabla de excepciones cuando un incremento salarial excede 10 por ciento**

El ejemplo 11.33 demuestra una limitación suave como alternativa a la limitación dura que ilustra el ejemplo 11.31. Debemos señalar que es preciso crear *LogTable* antes de crear el disparador. En Oracle, SEQUENCE es un objeto que mantiene valores únicos. La expresión *LogSeq.NextVal* genera el siguiente valor de la secuencia.

```

-- Create exception table and sequence
CREATE TABLE LogTable
(ExcNo      INTEGER PRIMARY KEY,
ExcTrigger  VARCHAR2(25) NOT NULL,
ExcTable    VARCHAR2(25) NOT NULL,
ExcKeyValue VARCHAR2(15) NOT NULL,
ExcDate     DATE DEFAULT SYSDATE NOT NULL,
ExcText     VARCHAR2(255) NOT NULL );
CREATE SEQUENCE LogSeq INCREMENT BY 1;
CREATE OR REPLACE TRIGGER tr_FacultySalary_UA
-- This trigger inserts a row into LogTable when
-- when a raise exceeds 10%.
AFTER UPDATE OF FacSalary
ON Faculty
FOR EACH ROW
WHEN (NEW.FacSalary > 1.1 * OLD.FacSalary)
DECLARE
    SalaryIncreaseTooHigh EXCEPTION;
    ExMessage VARCHAR(200);
BEGIN

```

```

        RAISE SalaryIncreaseTooHigh;
EXCEPTION
    WHEN SalaryIncreaseTooHigh THEN
        INSERT INTO LogTable
            (ExcNo, ExcTrigger, ExcTable, ExcKeyValue, ExcDate, ExcText)
        VALUES (LogSeq.NextVal, 'TR_FacultySalary_UA', 'Faculty',
                to_char(:New.FacSSN), SYSDATE,
                'Salary raise greater than 10%');

    END;
/
SET SERVEROUTPUT ON;
-- Test case 1: salary increase of 5%
UPDATE Faculty
    SET FacSalary = FacSalary * 1.05
    WHERE FacSSN = '543-21-0987';
SELECT FacSalary FROM Faculty WHERE FacSSN = '543-21-0987';
SELECT * FROM LogTable;

-- Test case 2: salary increase of 20% should generate an exception.
UPDATE Faculty
    SET FacSalary = FacSalary * 1.20
    WHERE FacSSN = '543-21-0987';
SELECT FacSalary FROM Faculty WHERE FacSSN = '543-21-0987';
SELECT * FROM LogTable;
ROLLBACK;

```

### 11.3.3 Cómo entender la ejecución de un disparador

Como se demostró en la sección anterior, por lo general los disparadores individuales son fáciles de entender. Sin embargo, en conjunto, pueden ser difíciles de entender sobre todo si están combinados con restricciones de integridad y acciones de base de datos. Para comprender el comportamiento colectivo de los disparadores, las restricciones de integridad y las acciones de manipulación de base de datos, es necesario conocer el procedimiento de ejecución que utiliza un DBMS. Aunque SQL:2003 especifica el procedimiento de ejecución de un disparador, la mayoría de los DBMS no se apegan a éste en forma estricta. Por tanto, esta subsección enfatiza el procedimiento de ejecución de disparadores en Oracle con comentarios de las diferencias entre los procedimientos de Oracle y SQL:2003.

#### procedimiento de ejecución de un disparador

especifica el orden de la ejecución entre distintos tipos de disparadores, restricciones de integridad y sentencias de manipulación de bases de datos. Los procedimientos de ejecución de disparadores pueden ser complejos porque las acciones de un disparador podrían accionar otros disparadores.

#### *Procedimiento simplificado de ejecución de disparadores*

El procedimiento de ejecución de disparadores se aplica a los sentencias de manipulación de datos (INSERT, UPDATE y DELETE). Antes de comenzar este procedimiento, Oracle determina los disparadores aplicables para una sentencia SQL. Un disparador es aplicable a un enunciado si contiene un evento que coincide con el tipo de sentencia. Para que una sentencia UPDATE coincida con una lista de columnas, por lo menos una de las columnas en el evento disparador debe estar en la lista de columnas actualizadas por la sentencia. Después de determinar los disparadores aplicables, Oracle ejecuta disparadores en el orden BEFORE STATEMENT, BEFORE ROW, AFTER ROW y AFTER STATEMENT. Un disparador aplicable no se ejecuta si su condición WHEN no es verdadera.

#### *Procedimiento simplificado de ejecución de disparadores en Oracle*

1. Ejecute los disparadores BEFORE STATEMENT aplicables.
2. Para cada fila afectada por la sentencia de manipulación SQL:

- 2.1. Ejecute los disparadores BEFORE ROW aplicables.
- 2.2. Realice la operación de manipulación de datos en la fila.
- 2.3. Lleve a cabo la revisión de la limitación de integridad.
- 2.4. Ejecute los disparadores AFTER ROW aplicables.
  
3. Lleve a cabo la revisión de la limitación de integración diferida.
4. Ejecute los disparadores de la sentencia AFTER aplicables.

El procedimiento de ejecución de disparadores de Oracle difiere un poco del de SQL:2003 para los disparadores superpuestos. Dos disparadores que tienen el mismo momento, granularidad y tabla aplicable se superponen si una sentencia SQL puede hacer que ambos se accionen. Por ejemplo, un disparador BEFORE ROW con el evento UPDATE ON Customer se superpone con un disparador BEFORE ROW con el evento UPDATE OF CustBal ON Customer. Ambos disparadores se accionan al actualizar la columna *CustBal*. Para los disparadores superpuestos, Oracle especifica que el orden de ejecución es arbitrario. Para SQL:2003, el orden de ejecución depende del momento en el que se define el disparador. Los disparadores superpuestos se ejecutan en el orden en que se crearon.

La superposición es útil para los disparadores UPDATE. Dos disparadores UPDATE en la misma tabla se pueden superponer aun cuando comprendan columnas diferentes. Por ejemplo, los disparadores UPDATE en *OffLocation* y *OffTime* se superponen si una sentencia UPDATE cambia en ambas columnas. Los disparadores no se superponen en las sentencias UPDATE que cambian sólo en una columna.

Como demostramos en el procedimiento simplificado de ejecución de disparadores, la mayor parte de la revisión de restricciones ocurre después de ejecutar los disparadores BEFORE ROW aplicables, pero antes de ejecutar los disparadores AFTER ROW aplicables. La revisión de restricciones diferida se lleva a cabo al final de una transacción. El capítulo 15, manejo de transacciones, presenta sentencias SQL para la revisión de limitación diferida. En la mayoría de las aplicaciones, pocas restricciones se declaran con revisión diferida.

#### *Procedimiento de ejecución de disparadores con ejecución recursiva*

Las sentencias de manipulación de datos en un disparador complican el procedimiento de ejecución simplificado. Las sentencias de manipulación de datos en un disparador pueden hacer accionar otros disparadores. Considere el disparador AFTER ROW en el ejemplo 11.29 que se acciona al agregar una fila a *Enrollment*. El disparador actualiza la columna *OffNumEnrolled* en la fila *Offering* relacionada. Supongamos que hay otro disparador en la columna *OffNumEnrolled* de la tabla *Offering* que se acciona cuando la columna *OffNumEnrolled* se alarga (digamos dos más allá del límite). Este segundo disparador se debe accionar como resultado de la activación del primero cuando un curso queda casi lleno.

Al encontrar una sentencia de manipulación de datos en un disparador, el procedimiento de ejecución de éste se realiza en forma recursiva. La ejecución recursiva significa que un procedimiento se llama a sí mismo. En el ejemplo anterior, el procedimiento de ejecución del disparador se ejecuta de manera recursiva al encontrar una sentencia de manipulación de datos en el disparador del ejemplo 11.29. En el procedimiento de ejecución de Oracle, los pasos 2.1 y 2.4 pueden comprender la ejecución recursiva del procedimiento. En el procedimiento de ejecución de SQL:2003, sólo el paso 2.4 puede comprender la ejecución recursiva porque SQL:2003 prohíbe las sentencias de manipulación de datos en los disparadores BEFORE ROW.

Las acciones en las filas de referencia también complican el procedimiento simplificado de ejecución. Al eliminar o actualizar una fila referenciada, la limitación de la llave externa puede especificar acciones (CASCADE, SET NULL y SET DEFAULT) en las filas relacionadas. Por ejemplo, una limitación de llave externa que contiene ON DELETE CASCADE para *Offering*. *CourseNo* significa que la eliminación de una fila *Course* provoca la eliminación de las filas relacionadas *Offering*. Las acciones en las filas referenciadas pueden hacer que se accionen otros disparadores dando lugar a una ejecución recursiva del procedimiento de ejecución de disparadores en el paso 2.3, tanto para Oracle como para SQL:2003. Las acciones en las filas de referencia se realizan como parte de la revisión de restricciones en el paso 2.3.

### **disparadores superpuestos**

dos o más disparadores con el mismo momento, granularidad y tabla aplicable. Los disparadores se superponen si una sentencia SQL puede hacer que ambos se accionen. Usted no debe depender de un orden de acción particular para los disparadores superpuestos.

A continuación, presentamos el procedimiento completo de ejecución de disparadores, con las complicaciones que causan una ejecución recursiva. La mayoría de los DBMS, como Oracle, limitan la profundidad de la ejecución recursiva en los pasos 2.1, 2.3 y 2.4.

#### *Procedimiento de ejecución de disparadores en Oracle con ejecución recursiva*

1. Ejecutar los disparadores BEFORE STATEMENT aplicables.
2. Para cada fila que afecte la sentencia de manipulación SQL:
  - 2.1. Ejecute los disparadores BEFORE ROW aplicables. Ejecute de manera recursiva el procedimiento para las sentencias de manipulación de datos en un disparador.
  - 2.2. Realice la operación de manipulación de datos en la fila.
  - 2.3. Lleve a cabo la revisión de la limitación de integridad. Ejecute en forma recursiva el procedimiento para las acciones en las filas referenciadas.
  - 2.4. Ejecute los disparadores AFTER ROW aplicables. Ejecute en forma recursiva el procedimiento para las sentencias de manipulación de datos en un disparador.
3. Lleve a cabo la revisión de las restricciones de integridad diferida.
4. Ejecute los disparadores aplicables de la sentencia AFTER.

El procedimiento completo de ejecución muestra una complejidad considerable al ejecutar un disparador. Para controlar la complejidad entre un conjunto de disparadores, deberá seguir estos lineamientos:

- Evite las sentencias de manipulación de datos en los disparadores BEFORE.
- Limite las sentencias de manipulación de datos en los disparadores AFTER a aquellos que es probable que tengan éxito.
- Para los disparadores que se accionan en sentencias UPDATE, siempre mencione las columnas en las que se aplica el disparador.
- Asegúrese de que los disparadores superpuestos no dependan de un orden de acción específico. En casi todos los DBMS, el orden de acción es arbitrario. Aun cuando no lo sea (como en SQL:2003), es riesgoso depender de un orden específico de acción.
- Tenga cuidado con los disparadores en las tablas afectadas por las acciones en las filas de referencia. Estos disparadores se van a accionar como resultado de las acciones en las tablas madre.

#### *Errores en tablas mutantes*

Oracle tiene una restricción para la ejecución de disparadores que pueden impedir el desarrollo de disparadores especializados. En cuanto a las acciones de los disparadores, Oracle prohíbe las sentencias SQL en la tabla en la que se define el disparador o en tablas relacionadas afectadas por acciones DELETE CASCADE. La tabla subyacente de disparadores y las tablas relacionadas se conocen como tablas mutantes. Por ejemplo, en un disparador para la tabla *Registration*, Oracle prohíbe sentencias SQL en las tablas *Registration* y *Enrollment*, si esta última contiene una limitación de clave externa en *Registration.RegNo* con la acción ON DELETE CASCADE. Si un disparador ejecuta un sentencia SQL en una tabla mutante, ocurre un error de tiempo de ejecución.

Para la mayoría de los disparadores, es posible evitar los errores en las tablas mutantes utilizando disparadores de fila con valores nuevos y antiguos. En situaciones especiales, es necesario volver a diseñar un disparador para evitar un error de tabla mutante. Una situación comprende un disparador para implementar una limitación de integridad que involucra otras filas de la misma tabla. Por ejemplo, un disparador para asegurar que no más de cinco filas tienen el mismo valor para una columna tendría un error de tabla mutante. Otro ejemplo sería un disparador que asegure que un renglón no se puede eliminar si es el último renglón relacionado con una tabla madre. Una segunda situación comprende un disparador para una tabla madre que inserta renglones en una tabla hija si ésta tiene una limitación de llave externa con ON DELETE CASCADE.

Para escribir disparadores en estas situaciones necesita una solución más compleja. Para detalles completos, debe consultar los sitios web que muestran soluciones para evitar errores de tablas mutantes. La documentación de Oracle menciona los dos enfoques siguientes:

1. Escribir un paquete y un conjunto de disparadores que utilice procedimientos en el paquete. Este último mantiene un arreglo privado que contiene los valores antiguos y nuevos de la tabla mutante. Por lo general, es necesario un disparador BEFORE STATEMENT para iniciar el arreglo privado, un disparador AFTER ROW para insertar en el arreglo privado y un disparador AFTER STATEMENT para implementar la limitación de integridad utilizando el arreglo privado.
2. Crear una vista y usar un disparador INSTEAD OF para ésta. Los disparadores de vista no tienen ninguna limitación para las tablas mutantes.

## Reflexión final

Este capítulo amplió su conocimiento sobre el desarrollo de aplicaciones de bases de datos con detalles acerca de los lenguajes de programación de base de datos, procedimientos almacenados y disparadores. Los lenguajes de programación de base de datos son lenguajes procedurales con una interfaz para uno o más DBMS. Ofrecen soporte para personalización, procesamiento por lotes y operaciones complejas más allá de la sentencia SQL SELECT, así como mayor eficiencia y portabilidad, en algunos casos. Los principales aspectos del diseño en un lenguaje de programación de base de datos son el estilo de lenguaje, unión, conexiones con bases de datos y procesamiento de resultados. Este capítulo presentó los antecedentes de PL/SQL, un lenguaje de programación de base de datos que se utiliza con mucha frecuencia y que está disponible como parte de Oracle.

Después de aprender sobre estos lenguajes y PL/SQL, este capítulo presentó los procedimientos almacenados. Estos procedimientos ofrecen módulos como procedimientos de lenguaje de programación. Los procedimientos almacenados que maneja un DBMS ofrecen beneficios adicionales que incluyen la posibilidad de volver a usar los planes de acceso, el manejo de dependencias y el control de la seguridad por parte del DBMS. Aprendió sobre la codificación de procedimientos en PL/SQL a través de ejemplos que muestran los procedimientos, funciones, procesamiento de excepciones y SQL integrado que contiene resultados de una sola fila y de varias filas con cursos. Además, aprendió sobre los paquetes PL/SQL que agrupan procedimientos, funciones y otros objetos PL/SQL relacionados.

La parte final del capítulo cubre los disparadores para el procesamiento de reglas empresariales. Un disparador comprende un evento, una condición y una secuencia de acciones. Aprendió los distintos usos para los disparadores, así como su clasificación según su granularidad, momento y eventos aplicables. Después de estos antecedentes, aprendió sobre la codificación de los disparadores Oracle utilizando sentencias PL/SQL en el cuerpo de un disparador. Para entender la complejidad de los conjuntos de disparadores muy numerosos, aprendió sobre los procedimientos de ejecución de disparadores especificando el orden de la ejecución entre distintos tipos de disparadores, restricciones de integridad y sentencias SQL.

El material en este capítulo es importante tanto para los desarrolladores de aplicaciones como para los administradores de bases de datos. Los procedimientos almacenados y disparadores pueden ser parte importante de aplicaciones extensas, quizás hasta 25 por ciento del código. Los desarrolladores de aplicaciones utilizan los lenguajes de programación de base de datos para codificar los procedimientos almacenados y disparadores, mientras que los administradores de bases de datos prestan atención al proceso de desarrollo. Además, los administradores de bases de datos pueden escribir procedimientos almacenados y disparadores para soportar el proceso de monitoreo o supervisión de bases de datos. Por tanto, los lenguajes de programación de base de datos, los procedimientos almacenados y los disparadores son herramientas importantes para las carreras de desarrollo de aplicaciones y administración de bases de datos.

## Revisión de conceptos

- La motivación primaria para los lenguajes de programación de base de datos: personalización, procesamiento en lotes y operaciones complejas.

- Motivación secundaria para los lenguajes de programación de base de datos: eficiencia y portabilidad.
- Interfaz en el nivel de sentencias para soportar SQL integrado en un lenguaje de programación.
- Interfaz en el nivel de llamadas para proporcionar procedimientos con el fin de invocar sentencias SQL en un lenguaje de programación.
- Popularidad de las interfaces en el nivel de llamadas patentadas (ODBC y JDBC), en lugar de la interfaz en el nivel de llamadas de SQL:2003.
- Soporte para unión estática y dinámica de sentencias SQL en interfaces a nivel de sentencias.
- Soporte para unión dinámica con posibilidad de volver a usar el plan de acceso para ejecuciones repetitivas en interfaces en el nivel de llamadas.
- Conexiones de bases de datos implícitas *versus* explícitas.
- Uso de cursos para integrar el procesamiento de un conjunto a la vez de SQL con el procesamiento de un registro a la vez de los lenguajes de programación.
- Tipos de datos PL/SQL y declaración de variables.
- Declaración de variables ancladas en PL/SQL.
- Sentencias condicionales en PL/SQL: IF-THEN, IF-THEN-ELSE, IF-THEN-ELSIF y CASE.
- Sentencias de ciclo en PL/SQL: FOR LOOP, WHILE LOOP y LOOP con una sentencia EXIT.
- Bloques anónimos para ejecutar sentencias PL/SQL y probar los procedimientos almacenados y disparadores.
- Motivaciones para los procedimientos almacenados: recopilación de planes de acceso, flexibilidad en el desarrollo cliente-servidor, implementación de operadores complejos y administración conveniente utilizando herramientas DBMS para el control de la seguridad y el manejo de dependencias.
- Especificaciones de los parámetros en los procedimientos y funciones PL/SQL.
- Procesamiento de excepciones en los procedimientos y funciones PL/SQL.
- Uso de cursos estáticos en los procedimientos y funciones PL/SQL.
- Cursos implícitos *versus* explícitos en PL/SQL.
- Paquetes PL/SQL para agrupar procedimientos, funciones y otros objetos relacionados.
- Especificación pública de paquetes *versus* especificación privada.
- Motivación para los disparadores: restricciones de integridad compleja, restricciones de transición, propagación de actualizaciones, reporte de excepciones y registro de auditorías.
- Granularidad de los disparadores: disparadores en el nivel de sentencias *versus* filas.
- Momento de los disparadores: antes y después de un evento.
- Eventos de los disparadores: INSERT, UPDATE o DELETE, así como eventos compuestos con combinaciones de éstos.
- Especificación de disparadores en SQL:2003 *versus* sintaxis de disparadores propietarios.
- Disparadores BEFORE ROW para restricciones de integridad compleja, restricciones de transición y estandarización de entrada de datos.
- Disparadores AFTER ROW de Oracle para la propagación de actualizaciones y reporte de excepciones.
- El orden de ejecución de los disparadores en un procedimiento de ejecución de disparadores: BEFORE STATEMENT, BEFORE ROW, AFTER ROW, AFTER STATEMENT.
- Orden de la implementación de restricciones de integridad en un procedimiento de ejecución de disparadores.
- Orden de ejecuciones arbitrarias para los disparadores superpuestos.
- Ejecución recursiva de un procedimiento de ejecución de disparadores para sentencias de manipulación de datos en el cuerpo de un disparador y acciones en las filas de referencia.

## Preguntas

1. ¿Qué es un lenguaje de programación de base de datos?
2. ¿Por qué la personalización es una motivación importante para los lenguajes de programación de base de datos?
3. ¿De qué manera los lenguajes de programación de base de datos ofrecen soporte para la personalización?
4. ¿Por qué el procesamiento en lotes es una motivación importante para los lenguajes de programación de base de datos?
5. ¿Por qué el soporte para operaciones complejas es una motivación importante para los lenguajes de programación de base de datos?
6. ¿Por qué la eficiencia es una motivación secundaria y no primaria para los lenguajes de programación de base de datos?
7. ¿Por qué la portabilidad es una motivación secundaria y no primaria para los lenguajes de programación de base de datos?
8. ¿Qué es una interfaz en el nivel de sentencias?
9. ¿Qué es una interfaz en el nivel de llamadas?
10. ¿Qué es la unión para un lenguaje de programación de base de datos?
11. ¿Cuál es la diferencia entre unión dinámica y unión estática?
12. ¿Qué relación existe entre el estilo del lenguaje y la unión?
13. ¿Qué sentencias y procedimientos de SQL:2003 ofrecen soporte para las conexiones explícitas de bases de datos?
14. ¿Qué diferencias es necesario resolver para procesar los resultados de una sentencia SQL en un programa de computadora?
15. ¿Qué es un cursor?
16. ¿Qué sentencias y procedimientos ofrece SQL:2003 para el procesamiento de cursos?
17. ¿Para qué estudiar el PL/SQL?
18. Explique la distinción entre mayúsculas y minúsculas en PL/SQL. ¿Por qué la mayor parte de los elementos distinguen entre mayúsculas y minúsculas?
19. ¿Qué es una declaración de variable anclada?
20. ¿Qué es una expresión lógica?
21. ¿Qué sentencias condicionales proporciona PL/SQL?
22. ¿Qué sentencias de iteración proporciona PL/SQL?
23. ¿Para qué se usa un bloque anónimo?
24. ¿Por qué un DBMS debe manejar procedimientos almacenados en lugar de un ambiente de programación?
25. ¿Cuáles son los usos de un parámetro en un procedimiento almacenado?
26. ¿Cuál es la restricción en el tipo de datos en la especificación de un parámetro?
27. ¿Para qué utilizar excepciones predefinidas y excepciones definidas por el usuario?
28. ¿Para qué usar la excepción OTHERS?
29. ¿En qué se diferencia una función de un procedimiento?
30. ¿Cuáles son las dos clases de declaración de cursor en PL/SQL?
31. ¿Cuál es la diferencia entre un cursor estático y uno dinámico en PL/SQL?
32. ¿Qué es un atributo de cursor?
33. ¿Cómo se hace referencia a los atributos de cursor?
34. ¿Cuál es el propósito de un paquete PL/SQL?
35. ¿Para qué separar la interfaz de la implementación en un paquete PL/SQL?
36. ¿Qué contiene una interfaz de paquete?
37. ¿Qué contiene un paquete de implementación?
38. ¿Qué es un nombre alternativo para un disparador?
39. ¿Cuáles son los usos típicos de los disparadores?
40. ¿Cómo clasifica SQL:2003 a los disparadores?
41. ¿Por qué la mayoría de las implementaciones de disparadores difieren de la especificación de SQL:2003?

42. ¿De qué manera se especifican los eventos compuestos en un disparador?
43. ¿Cómo se prueban los disparadores?
44. ¿Es preferible escribir muchos disparadores pequeños o pocos disparadores grandes?
45. ¿Qué es un procedimiento de ejecución de disparadores?
46. ¿Cuál es el orden de ejecución de diversos tipos de disparadores?
47. ¿Qué es un disparador superpuesto? ¿Cuál es el orden de ejecución de los disparadores superpuestos?
48. ¿Qué situaciones llevan a la ejecución recursiva del procedimiento de ejecución de disparadores?
49. Mencione por lo menos dos formas de reducir la complejidad de un conjunto de disparadores.
50. ¿Qué es un error de tabla mutante en un disparador de Oracle?
51. ¿Cómo se evitan los errores de tabla mutante?
52. ¿Cuáles son los usos típicos de los disparadores BEFORE ROW?
53. ¿Cuáles son los usos típicos de los disparadores AFTER ROW?
54. ¿Cuál es la diferencia entre una restricción dura y una suave?
55. ¿Qué clase de disparador se puede escribir para implementar una restricción suave?
56. ¿En qué se diferencia el procedimiento de ejecución de disparadores de Oracle del procedimiento de SQL:2003 para ejecuciones recursivas?

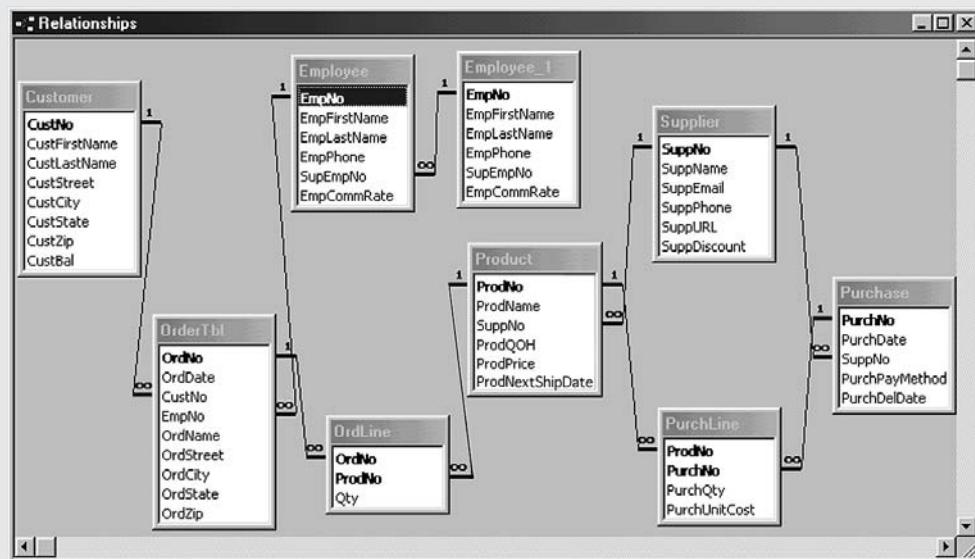
## Problemas

Cada problema usa la base de datos de captura de pedidos revisada que mostramos en el capítulo 10. Para su consulta, la figura 11.P1 muestra una ventana de relación para esta base de datos. En los problemas del capítulo 10 encontrará más detalles sobre la base de datos revisada.

Los problemas le ayudan a practicar en la codificación PL/SQL y el desarrollo de procedimientos, funciones, paquetes y disparadores. Además, algunos problemas comprenden bloques anónimos y guiones para probar los procedimientos, funciones, paquetes y disparadores.

1. Escriba un bloque anónimo PL/SQL para calcular el número de días en un año no bisiesto. Su código debe completar un ciclo con los meses del año (1 al 12) utilizando una sentencia FOR LOOP. Deberá usar una sentencia IF-THEN-ELSIF para determinar el número de días que deberá agregar al mes. Puede agrupar los meses que tienen el mismo número de días. Muestre el número de días antes de que se complete el ciclo.
2. Revise el problema 1 para calcular el número de días en un año bisiesto. Si trabaja en Oracle 9i o posterior, utilice una sentencia CASE en lugar de uno IF-THEN-ELSIF. Recuerde que no puede usar la sentencia CASE en Oracle 8i.
3. Escriba un bloque anónimo PL/SQL para calcular el valor futuro de 1,000 dólares con un interés de 8 por ciento, compuesto anualmente durante 10 años. El valor futuro al final del año  $i$  es la cantidad al principio del año más la cantidad inicial por la tasa de interés anual. Utilice WHILE LOOP para calcular el valor futuro. Muestre la cantidad futura antes de cerrar el ciclo.

**FIGURA 11.P1**  
Diagrama de relación para la base de datos captura de pedidos revisada



4. Escriba un bloque anónimo PL/SQL para mostrar el precio del producto número P0036577. Utilice una declaración de variable anclada y una sentencia SELECT INTO para determinar el precio. Si éste es menor que 100 dólares, muestre un mensaje de que el producto es una buena compra. Si el precio es entre 100 y 300 dólares, presente un mensaje de que el producto tiene un precio competitivo. Si el precio es mayor que 300 dólares, muestre un mensaje de que el producto ofrece numerosas características.
5. Escriba un procedimiento PL/SQL para insertar un nuevo renglón en la tabla *Product* utilizando parámetros de entrada para el número de producto, nombre de producto, precio de producto, fecha del próximo envío, cantidad disponible y número de proveedor. Para una inserción exitosa, muestre un mensaje apropiado. Si ocurre un error en la sentencia INSERT, cree una excepción con un mensaje de error apropiado.
6. Revise el problema 5 para generar un valor de salida en lugar de desplegar un mensaje sobre una inserción exitosa. Además, el procedimiento revisado debe detectar un error de llave primaria duplicada. Si el usuario trata de insertar un renglón con un número de producto existente, su procedimiento debe crear una excepción con un mensaje de error apropiado.
7. Escriba guiones de prueba para los procedimientos en los problemas 5 y 6. Para el procedimiento en el problema 6, su guión debe realizar una prueba para detectar una violación a la llave primaria y una violación a una llave externa.
8. Escriba una función PL/SQL para determinar si el pedido más reciente para un número de cliente determinado se envió a la dirección de facturación del cliente. La función debe regresar TRUE si cada una de las columnas de dirección de pedido (calle, ciudad, estado y código postal) es igual a la columna de dirección de cliente correspondiente. Si cualquiera de las columnas de dirección no es igual, el resultado será FALSE. El pedido más reciente tiene la fecha de pedido más larga. El resultado será NULL si el cliente no existe o no hay pedidos para él.
9. Cree un *script* de prueba para la función PL/SQL en el problema 8.
10. Cree un procedimiento para calcular la comisión para un número de pedido determinado. La comisión es la tasa de comisión del empleado que toma el pedido por la cantidad del pedido. La cantidad de un pedido es la suma del producto de la cantidad del producto ordenado por el precio del producto. Si el pedido no tiene un empleado relacionado (se hizo en la web), la comisión es cero. El procedimiento debe tener una variable de salida para la cantidad de la comisión. La variable de salida debe ser nula si no existe un pedido.
11. Cree un *script* de prueba para el procedimiento PL/SQL en el problema 10.
12. Cree una función para revisar la cantidad disponible de un producto. Los parámetros de entrada son un número de producto y una cantidad ordenada. El resultado será FALSE si la cantidad disponible es menor que la cantidad ordenada. El resultado será TRUE si la cantidad disponible es mayor o igual que la cantidad ordenada. El resultado será NULL si el producto no existe.
13. Cree un procedimiento para insertar una línea de pedido. Utilice la función del problema 12 para revisar que haya existencias adecuadas. Si no hay existencias suficientes, el parámetro de salida deberá ser FALSE. Cree una excepción si hay un error de inserción como una llave primaria duplicada.
14. Cree *scripts* de prueba para la función en el problema 12 y el procedimiento en el problema 13.
15. Escriba una función para calcular la mediana de la columna saldo de clientes. La mediana es el valor intermedio en una lista de números. Si el tamaño de la lista es par, la mediana es el promedio de los dos valores intermedios. Por ejemplo, si hay 18 saldos de clientes, la mediana es el promedio de los saldos noveno y décimo. Debe usar un cursor implícito en su función. Para escribir esta función tal vez desee utilizar las funciones SQL *Trunc* y *Mod* de Oracle. Escriba un *script* de prueba para su función. Cabe señalar que esta función no tiene ningún parámetro. No utilice paréntesis en la declaración ni en la invocación de la función cuando ésta no tenga parámetros.
16. Revise la función del problema 15 con un cursor explícito utilizando las sentencias CURSOR, OPEN, FETCH y CLOSE. Escriba un *script* de prueba para la función revisada.
17. Cree un paquete que contenga la función del problema 15, el procedimiento del problema 13, el procedimiento del problema 10, la función del problema 8 y el procedimiento del problema 6. La función del problema 12 debe ser privada para el paquete. Escriba un *script* de prueba para ejecutar cada objeto público en el paquete. No necesita probar cada objeto público por completo. Una ejecución por objeto público está bien porque anteriormente probó los procedimientos y funciones fuera del paquete.
18. Escriba un disparador AFTER ROW para que se active con cada acción en la tabla *Customer*. En el disparador, muestre los valores de clientes nuevos y antiguos cada vez que el disparador se active. Escriba un *script* para probar el disparador.

19. Escriba un disparador para la limitación de transición en la tabla *Employee*. El disparador debe evitar actualizaciones que aumenten o disminuyan la tasa de comisión más de 10 por ciento. Escriba un *script* para probar su disparador.
20. Escriba un disparador para eliminar el prefijo http:// en la columna *Supplier.SuppURL* en las operaciones de inserción y actualización. Su disparador deberá funcionar sin importar el caso del prefijo http://. Debe usar funciones SQL de Oracle para la manipulación de hileras. Tiene que estudiar las funciones SQL de Oracle como *SubStr*, *Lower* y *LTrim*. Escriba un *script* para probar su disparador.
21. Escriba un disparador para asegurarse que hay existencias adecuadas al insertar una nueva fila *OrdLine* o al actualizar la cantidad de la fila *OrdLine*. En las operaciones de inserción, *ProdQOH* de la fila relacionada debe ser mayor o igual a la cantidad en la nueva fila. En las operaciones de actualización, *ProdQOH* debe ser mayor o igual a la diferencia (cantidad nueva menos cantidad antigua).
22. Escriba un disparador para propagar actualizaciones a la tabla *Product* después de una operación en la tabla *OrdLine*. Para inserciones, el disparador debe reducir la cantidad disponible según la cantidad del pedido. Para actualizaciones, el disparador debe reducir la cantidad disponible según la diferencia entre la nueva cantidad del pedido y la antigua cantidad del pedido. Para eliminaciones, el disparador debe incrementar la cantidad disponible de acuerdo con la cantidad antigua del pedido.
23. Escriba un *script* para probar los disparadores de los problemas 21 y 22.
24. Escriba un disparador para propagar las operaciones de inserción en la tabla *PurchLine*. La cantidad disponible debe aumentar de acuerdo con la cantidad de la compra. Escriba un *script* para probar el disparador.
25. Escriba un disparador para propagar las actualizaciones a la tabla *Product* después de las operaciones de actualización de la tabla *PurchLine*. La cantidad disponible debe aumentar de acuerdo con la diferencia entre la cantidad de la compra nueva y la cantidad de la compra antigua. Escriba un *script* para probar el disparador.
26. Escriba un disparador para propagar las actualizaciones a la tabla *Product* después de las operaciones de eliminación en la tabla *PurchLine*. La cantidad disponible debe disminuir de acuerdo con la cantidad de la compra antigua. Escriba un *script* para probar el disparador.
27. Escriba un disparador para propagar las actualizaciones a la tabla *Product* en la columna *ProdNo* de la tabla *PurchLine*. La cantidad disponible del producto antiguo debe disminuir, mientras que la cantidad disponible del producto nuevo debe aumentar. Escriba un *script* para probar el disparador.
28. Suponga que tiene una sentencia UPDATE que cambia tanto la columna *ProdNo* como la columna *PurchQty* de la tabla *PurchLine*. ¿Qué disparadores (de los que escribió en problemas anteriores) se activan con esa sentencia UPDATE? Si se activa más de uno, ¿por qué los disparadores se superponen y en qué orden se activan? Modifique los disparadores superpuestos y prepare un *script* de prueba para poder determinar el orden de activación. ¿El procedimiento de ejecución de disparadores de Oracle garantiza el orden de activación?
29. Para la sentencia UPDATE en el problema 28, ¿los disparadores que creó en problemas anteriores funcionan correctamente? Escriba un *script* para probar sus disparadores en la sentencia UPDATE. Si los disparadores no funcionan de manera correcta, vuelva a escribirlos de modo que funcionen para la sentencia UPDATE en ambas columnas, así como en las columnas individuales. Escriba un *script* para probar los disparadores revisados. *Consejo:* Necesita especificar la columna en la palabra clave UPDATING en el cuerpo del disparador. Por ejemplo, puede especificar *UPDATING('PurchQty')* para verificar si se actualizó al columna *PurchQty*.
30. ¿Puede idear otra solución para el problema de las sentencias UPDATE que cambian las columnas *ProdNo* y *PurchQty*? ¿Es razonable que las aplicaciones en línea ofrezcan soporte para esas sentencias UPDATE?
31. Escriba un disparador para implementar una limitación dura en la columna *Product.ProdPrice*. El disparador debe evitar las actualizaciones que aumenten o disminuyan el valor más de 15 por ciento. Escriba un *script* para probar el disparador.
32. Escriba un disparador para implementar una limitación suave en la columna *Product.ProdPrice*. El disparador debe insertar una fila en una tabla de excepciones para las actualizaciones que aumenten o disminuyan el valor más de 15 por ciento. Utilice la tabla de excepciones que se muestra en el ejemplo 11.33. Escriba un *script* para probar el disparador.

## Referencias para ampliar su estudio

*Oracle Technology Network* ([www.oracle.com/technology](http://www.oracle.com/technology)) contiene gran cantidad de material acerca de PL/SQL, procedimientos almacenados y disparadores. *PL/SQL User's Guide* ofrece detalles sobre PL/SQL y procedimientos almacenados. *Oracle SQL Reference* proporciona detalles sobre los disparadores, así

como descripciones de funciones predefinidas como *Mod* y *SubStr*. Encontrará más detalles y ejemplos en *Oracle Concepts* y *Oracle Application Developers Guide*. Melton y Simon (2001) describen disparadores en SQL:1999.

## Apéndice 11.A

### Resumen de la sintaxis de SQL:2003

Este apéndice resume la sintaxis de SQL:2003 para la sentencia disparador. Las convenciones utilizadas en la sintaxis son idénticas a aquellas que se usan al final del capítulo 3.

#### Sentencia disparador

```

CREATE TRIGGER TriggerName
  <TriggerTiming> <TriggerEvent> ON TableName
  [ REFERENCING <AliasClause> [ <AliasClause> ] ]
  [ <GranularityClause> [ WHEN ( <Row-Condition> ) ] ]
  <ActionBlock>

<TriggerTiming>: { BEFORE | AFTER }
<TriggerEvent>: { INSERT | DELETE | UPDATE [ OF ColumnName* ] }
<AliasClause>: { <RowAlias> | <TableAlias> }
<RowAlias>:
  { OLD [ ROW ] [ AS ] AliasName | NEW [ ROW ] [ AS ] AliasName }

<TableAlias>:
  { OLD TABLE [AS] AliasName | NEW TABLE [AS] AliasName }

<GranularityClause>: FOR EACH { ROW | STATEMENT }

<Row-Condition>: -- defined in Chapter 3

<ActionBlock>:
  -- can be a procedure call or an SQL:2003 block

```



Parte

# Desarrollo avanzado de bases de datos

---

6

La parte 6 cubre los temas del desarrollo avanzado de bases de datos, para ampliar los conocimientos y habilidades adquiridos en las partes 2 a 5. La parte 6 es una sección que enfatiza la integración del material de los capítulos anteriores y el desarrollo de bases de datos para problemas de grandes empresas. El capítulo 12 describe el diseño e integración de vistas, conceptos de modelado de datos para esfuerzos de desarrollo de bases de datos extensas. El capítulo 13 ofrece un amplio estudio de caso que permite a los estudiantes darse una idea de las dificultades al aplicar el diseño de bases de datos y las habilidades de desarrollo de aplicaciones a una situación de negocios realista.

---

**Capítulo 12.** Diseño e integración de vistas

**Capítulo 13.** Desarrollo de bases de datos para Student Loan Limited



# Capítulo

# 12

---

## Diseño e integración de vistas

### Objetivos de aprendizaje

Este capítulo describe la práctica para diseñar vistas de usuario y combinarlas en un diseño conceptual completo. Al finalizar este capítulo, el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Entender la motivación para el diseño y la integración de vistas.
- Analizar un formulario y construir un ERD para representarlo.
- Determinar una estrategia de integración para un esfuerzo de desarrollo de bases de datos.
- Llevar a cabo planteamientos de integración incremental y paralela.
- Reconocer y resolver sinónimos y homónimos en el proceso de integración de vistas.

### Panorama general

---

Los capítulos 5, 6 y 7 proporcionan herramientas para el modelado y la normalización de datos, habilidades fundamentales para el diseño de bases de datos. Usted aplicó estos conocimientos en la construcción de diagramas de entidad-relación (ERD) para problemas medianos, convertir ERD en tablas de relación y normalizar las tablas. Este capítulo amplía sus habilidades para desarrollar bases de datos mostrando un planteamiento para analizar vistas e integrar vistas de usuario en un esquema conceptual completo. Este planteamiento está orientado hacia las aplicaciones apropiadas para diseñar bases de datos complejas.

Para convertirse en un buen diseñador de bases de datos, necesita ampliar sus habilidades hacia grandes problemas. Con el fin de motivarlo sobre la importancia de ampliar sus capacidades, este capítulo describe la naturaleza de grandes proyectos de desarrollo de bases de datos. Luego, presenta una metodología para diseñar vistas, poniendo énfasis en la construcción de un ERD para representar un formulario de captura de datos. Los formularios son importantes fuentes de requerimientos para el diseño de bases de datos. Va a aprender a analizar cada formulario, construir un ERD y verificar la consistencia del ERD con el formulario. Gracias al énfasis en las vistas y los formularios, este capítulo seguiría en forma lógica al capítulo 10, que trata del desarrollo de aplicaciones con vistas. Mientras estudia este capítulo tal vez quiera revisar importantes conceptos de dicho capítulo, como las vistas que se pueden actualizar.

Después de la presentación del diseño de vistas, este capítulo describe el proceso de integración de las vistas, combinando ERD que representan vistas individuales. Usted aprenderá sobre los planteamientos de integración incremental y paralela, la determinación de una estrategia de integración mediante el análisis de las relaciones entre el formulario y la aplicación del proceso de integración utilizando planteamientos incrementales y paralelos.

## 12.1 Motivación para el diseño e integración de vistas

La complejidad de una base de datos refleja la complejidad de la organización subyacente y las funciones que esa base de datos soporta. Existen muchos factores que contribuyen a la complejidad de una organización. Desde luego, el tamaño es un factor determinante de la complejidad y puede medirse de varias formas, como el volumen de ventas, número de empleados, número de productos y número de países en los que la organización opera. Sin embargo, el tamaño en sí no es el único factor determinante. Otros factores que contribuyen a la complejidad organizacional son el ambiente regulatorio, el ambiente competitivo y la estructura organizacional. Por ejemplo, las áreas de nómina y personal pueden ser muy complejas debido a la variedad de tipos de empleados, los diversos paquetes de compensaciones, los acuerdos sindicales y las regulaciones gubernamentales.

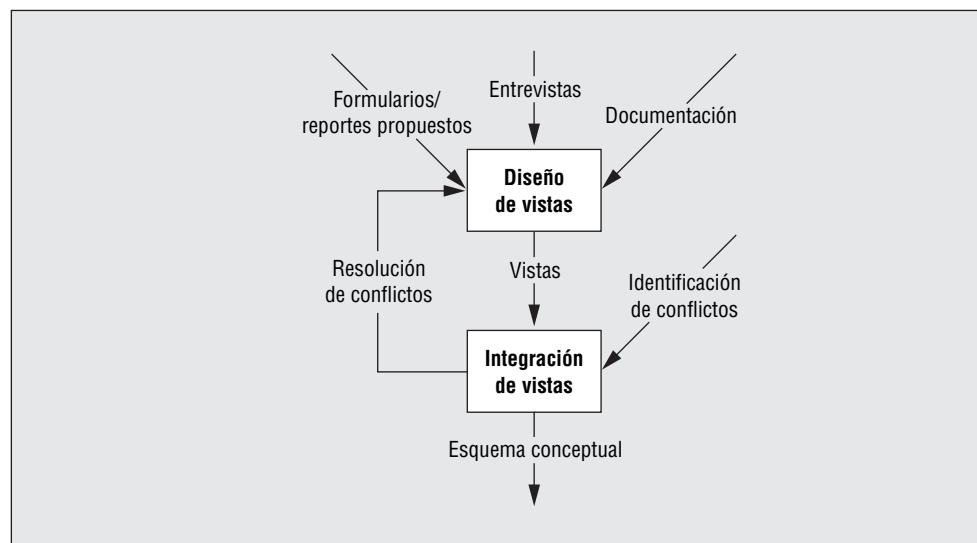
Las grandes organizaciones tienen muchas bases de datos con bases de datos individuales que soportan grupos de funciones como nómina, personal, contabilidad, requerimientos de materiales, etc. Estas bases de datos individuales suelen ser muy complejas, ya que se miden según el tamaño de los ERD. Un ERD para una base de datos extensa puede tener cientos de tipos de entidad-relación. Al convertirla en una base de datos relacional, la base de datos llega a presentar cientos o quizás miles de tablas. Un ERD grande es difícil de inspeccionar visualmente porque puede llenar toda una pared. Otras medidas de complejidad comprenden el uso de una base de datos a través de formularios, reportes, procedimientos almacenados y disparadores. Una base de datos extensa logra tener de cientos a miles de formularios, reportes, procedimientos almacenados y disparadores.

El diseño de bases de datos extensas es un proceso que consume mucho tiempo y requiere de mucho trabajo. El esfuerzo de diseño comprende la recopilación de requerimientos de muchos grupos de usuarios distintos. Los requerimientos pueden ser muy difíciles de capturar. Los usuarios necesitan experimentar un sistema para detectar sus requerimientos. Gracias al volumen de requerimientos y a su dificultad de captura, el esfuerzo de diseño de una base de datos muy grande puede requerir de un equipo de diseñadores. La coordinación entre los diseñadores es parte importante del esfuerzo de diseño de bases de datos.

Para manejar la complejidad, en muchas áreas de la computación se emplea la estrategia “divide y vencerás”. El hecho de dividir un problema grande permite solucionar los más pequeños de manera independiente. Luego, las soluciones a estos problemas se combinan en una sola para atender la totalidad.

El diseño y la integración de vistas (figura 12.1) ofrecen soporte para el manejo de la complejidad en el esfuerzo de diseño de bases de datos. En el diseño de vistas se construye un

**FIGURA 12.1**  
Panorama general del diseño e integración de vistas



ERD para cada grupo de usuarios. Los requerimientos pueden tener diversos formatos como entrevistas, documentación de un sistema existente y formularios y reportes propuestos. Por lo regular, una vista es tan pequeña que una sola persona puede diseñarla. Varios diseñadores pueden trabajar en vistas que cubren distintas partes de una base de datos. El proceso de integración de vistas combina las vistas en un esquema conceptual completo. La integración comprende el reconocimiento y la solución de conflictos. Para resolver conflictos es necesario revisar en ocasiones las vistas conflictivas. El compromiso es una parte importante de la resolución de conflictos en el proceso de integración.

Las secciones restantes de este capítulo proporcionan detalles sobre las actividades de diseño e integración de vistas. Enfatizamos sobre todo los formularios de captura de datos como una fuente de requisitos.

## 12.2 Diseño de vistas con formularios

Los formularios pueden ofrecer una fuente de requisitos importante para el diseño de bases de datos. Gracias a que les son familiares, los usuarios pueden comunicar con efectividad varios requerimientos a través de los formularios que utilizan. Para ayudarle en el uso de formularios como requerimientos de las bases de datos, esta sección describe un procedimiento para diseñar vistas usando formularios de captura de datos. El procedimiento le permite analizar los requerimientos de datos de un formulario. Después del procedimiento de análisis del formulario, estudiamos su aplicación en los formularios con relaciones M-way.

### 12.2.1 Análisis de formularios

Al utilizar formularios para el diseño de bases de datos se invierte el proceso tradicional del desarrollo de bases de datos. En éste, el diseño precede al desarrollo de aplicaciones. Con un planteamiento basado en formularios para el diseño de bases de datos, los formularios se definen antes o en el momento de diseñar la base. Los formularios pueden ser en formato de papel o como parte de un sistema ya existente. La definición del formulario no necesita ser tan completa como se requiere después de terminar el diseño de la base de datos. Por ejemplo, no es necesario definir todo el conjunto de eventos para la interacción del usuario con un formulario sino hasta las últimas etapas del proceso de desarrollo. En un principio, la definición del formulario puede comprender un boceto en un procesador de texto (figura 12.2) o una herramienta de dibujo. Además, quizás necesite varias muestras de un formulario.

**FIGURA 12.2**  
Muestra de un formulario de pedido de cliente

<b>Customer Order Form</b>			
<b>Order No.:</b> 1234	<b>Order Date:</b> 3/19/2006		
<b>Customer No.:</b> 1001	<b>Customer Name:</b> Jon Smith		
<b>Address.:</b> 123 Any Street			
<b>City:</b> Seattle	<b>State:</b> WA	<b>Zip:</b> 98115	
<b>Salesperson No.:</b> 1001		<b>Salesperson Name:</b> Jane Doe	
Product No.	Description	Quantity	Unit Price
M128	Bookcase	4	\$120
B138	Cabinet	3	\$150
R210	Table	1	\$500

← Madre  
(formulario principal)

← Hija  
(subformulario)

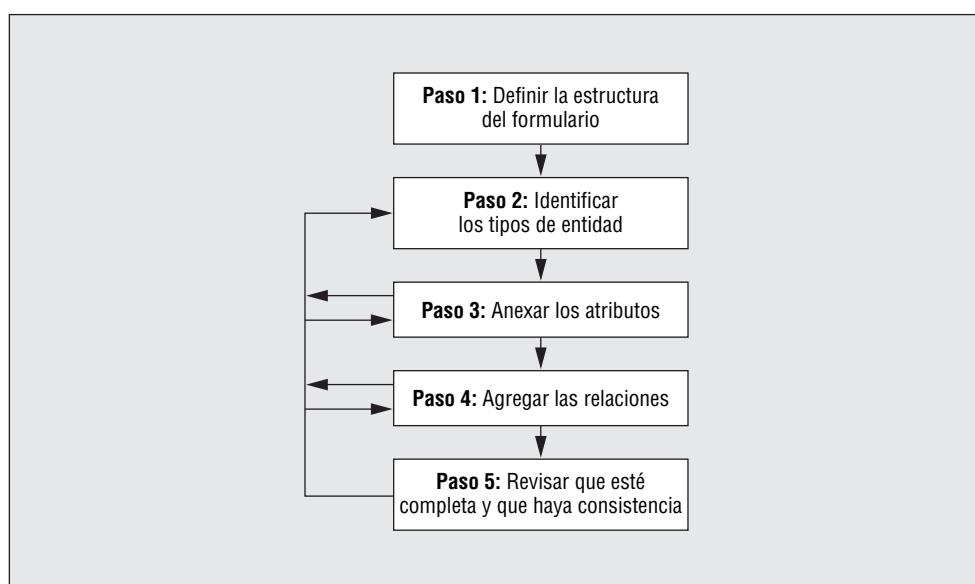
Debe utilizar todo tipo de requerimientos en el proceso de diseño de vistas. Los formularios deben analizarse detenidamente ya que son una importante fuente de requerimientos.

En el análisis de formularios (figura 12.3) puede crear un diagrama de entidad-relación para representar un formulario. El ERD resultante es una vista de la base de datos. El ERD debe ser general para soportar el formulario y otros procedimientos anticipados. El retroceso en la figura 12.3 muestra que el proceso de análisis de formularios puede regresar a los pasos anteriores. No es necesario seguir los pasos de manera secuencial. Si se presenta algún problema, particularmente en el último paso, es necesario repetir otros pasos para corregirlo. El resto de esta sección explica con más detalle los pasos del análisis de formularios y aplica este proceso en ejemplos de formularios.

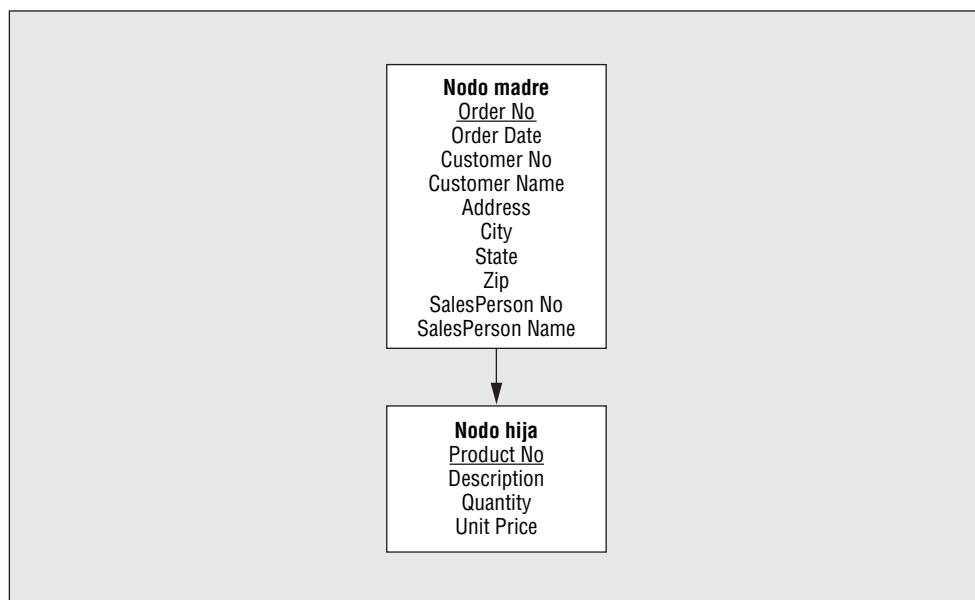
#### *Paso 1: Definir la estructura del formulario*

En el primer paso, usted construye una jerarquía que ilustra la estructura del formulario. La mayoría de los formularios consiste en una jerarquía sencilla en la que el formulario principal

**FIGURA 12.3**  
Pasos en el proceso  
de análisis de formu-  
larios



**FIGURA 12.4**  
Estructura jerárquica  
para el formulario de  
pedido de cliente



es la madre y el subformulario es la hija. Por ejemplo, la figura 12.4 ilustra la estructura del formulario de pedido de cliente de la figura 12.2. Un rectángulo (madre o hija) en el diagrama jerárquico se conoce como nodo. Los formularios complejos pueden tener más nodos (subformularios paralelos) y más niveles en la jerarquía (subformularios dentro de subformularios). Por ejemplo, un formulario de pedido de servicio automotriz puede tener un subformulario (hija) que muestra los cargos por refacciones y otro subformulario (hija) que muestra los cargos por mano de obra. Los formularios complejos, como el formulario de pedido de un servicio, no son tan comunes porque son difíciles de entender para los usuarios.

Como parte de la elaboración de la estructura del formulario, debe identificar las llaves en cada nodo de la jerarquía. En la figura 12.4, las llaves del nodo están subrayadas. En el nodo madre, el valor de llave de nodo es único entre todos los ejemplos de formulario. En el nodo hija, el valor de la llave de nodo es único en el nodo madre. Por ejemplo, un número de producto es único en un pedido. Sin embargo, dos pedidos pueden utilizar el mismo número de producto.

### *Paso 2: Identificar los tipos de entidad*

En el segundo paso, puede dividir cada nodo de la estructura jerárquica en uno o más tipos de entidad. Por lo general, cada nodo de la estructura jerárquica representa más de un tipo de entidad. Deberá buscar los campos de los formularios que pueden ser llaves primarias de un tipo de entidad en la base de datos. Deberá hacer un tipo de entidad si el campo del formulario es una llave primaria potencial y hay otros campos asociados en el formulario. De igual forma, debe agrupar los campos de formularios en tipos de entidad utilizando dependencias funcionales (FD). Todos los campos de formulario que determinan el(s) mismo(s) campo(s) se deben colocar juntos en el mismo tipo de entidad.

Como ejemplo del paso 2 hay tres tipos de entidad en el nodo madre de la figura 12.4, como muestra la figura 12.5: *Customer*, identificado por *Customer No*; *Order*, identificado por *Order No*, y *Salesperson*, identificado por *Salesperson No*. La llave del nodo madre (*Order No*) casi siempre indica un tipo de entidad. *Customer No* y *Salesperson No* son buenas opciones porque hay otros campos asociados (*Customer Name* y *Salesperson Name*). En el nodo hija, hay un tipo de entidad: *Product*, designado por *Product No*, porque *Product No* puede ser una llave primaria con otros campos asociados.

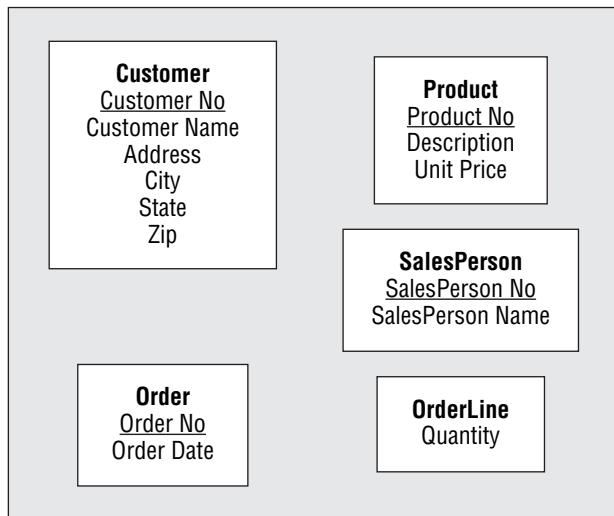
### *Paso 3: Anexar atributos*

En el tercer paso usted tiene que anexar los atributos a los tipos de identidad que identificó en el paso anterior. Casi siempre es fácil asociar los campos de los formularios con los tipos de entidad. Deberá agrupar los campos que están relacionados con las llaves primarias encontradas en el paso 2. En ocasiones, la cercanía de los campos proporciona algunas claves para agruparlos: con frecuencia los campos de los formularios que están cerca pertenecen al mismo tipo de entidad. En este ejemplo, agrupe los campos como muestra la figura 12.6. *Order* con *Order No* y *Order Date*; *Customer* con *Customer No*, *Customer Name*, *Address*, *City*, *State* y *Zip*; *Salesperson* con *Salesperson No* y *Salesperson Name*, y *Product* con *Product No*, *Description* y *Unit Price*.

**FIGURA 12.5**  
Tipos de entidad para el formulario de pedido de cliente



**FIGURA 12.6**  
Atributos agregados  
a los tipos de entidad  
de la figura 12.5



**TABLA 12.1**  
Reglas para conectar  
los tipos de entidad

1. Coloque el tipo de entidad del formulario en el centro del ERD.
2. Agregue las relaciones entre el tipo de entidad del formulario y otros tipos de entidad derivados del nodo madre. Por lo regular, las relaciones son 1-M.
3. Agregue una relación para conectar el tipo de entidad del formulario con un tipo de entidad en el nodo hija.
4. Agregue las relaciones para conectar los tipos de entidad derivados del nodo hija, si todavía no están conectados.

Si es observador quizás se dé cuenta de que *Quantity* no parece pertenecer a *Product* porque la combinación de *Order No* y *Product No* determina *Quantity*. Puede crear un nuevo tipo de entidad (*OrderLine*) con *Quantity* como atributo. Si le hace falta este tipo de entidad en el paso siguiente, puede convertir *Quantity* en un atributo de una relación. Además, al atributo *Unit Price* se le puede considerar un atributo del tipo de entidad *OrderLine* si se registra el precio histórico en lugar del precio actual de un producto.

#### Paso 4: Agregar relaciones

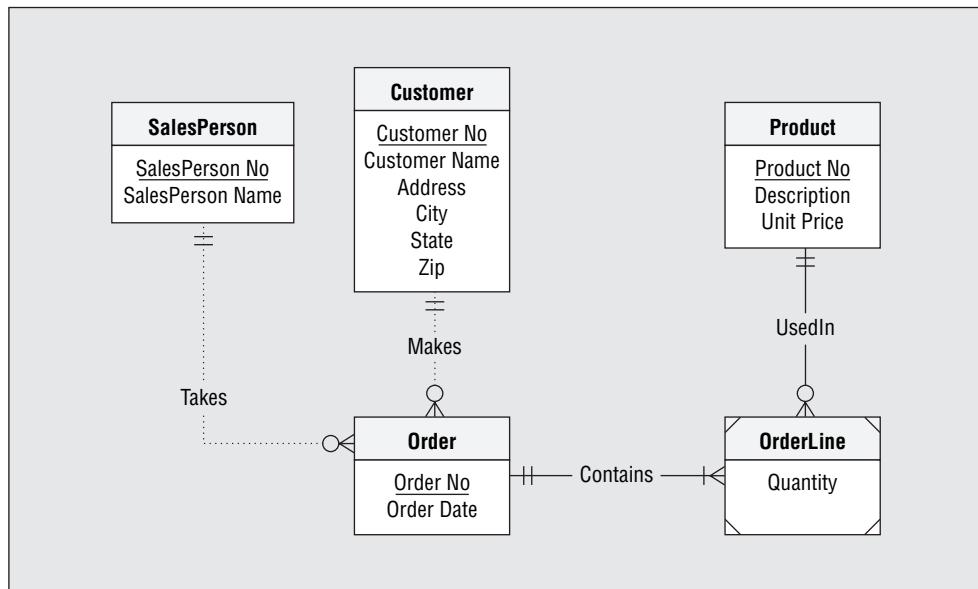
En el cuarto paso usted conecta los tipos de entidad con las relaciones y especifica las cardinalidades. La tabla 12.1 resume las reglas para conectar los tipos de entidad. Debe empezar con el tipo de entidad que contiene la llave primaria del formulario principal. Consideremos que éste es el tipo de entidad del formulario y hagamos que sea el centro del ERD. Por lo general, varias relaciones conectan el tipo de entidad del formulario con otros tipos de entidad derivados de los nodos madre e hija. En la figura 12.5, *Order* es el tipo de entidad del formulario.

Después de identificar el tipo de entidad del formulario, deberá agregar relaciones 1-M con otros tipos de entidad derivados de los campos en el formulario principal. Esto deja a *Order* conectado con *Customer* y *SalesPerson* a través de relaciones 1-M, como muestra la figura 12.7. Deberá verificar que el mismo cliente pueda hacer varios pedidos y el mismo vendedor pueda tomar varios pedidos examinando ejemplos de formularios adicionales y hablando con usuarios que tengan amplios conocimientos.

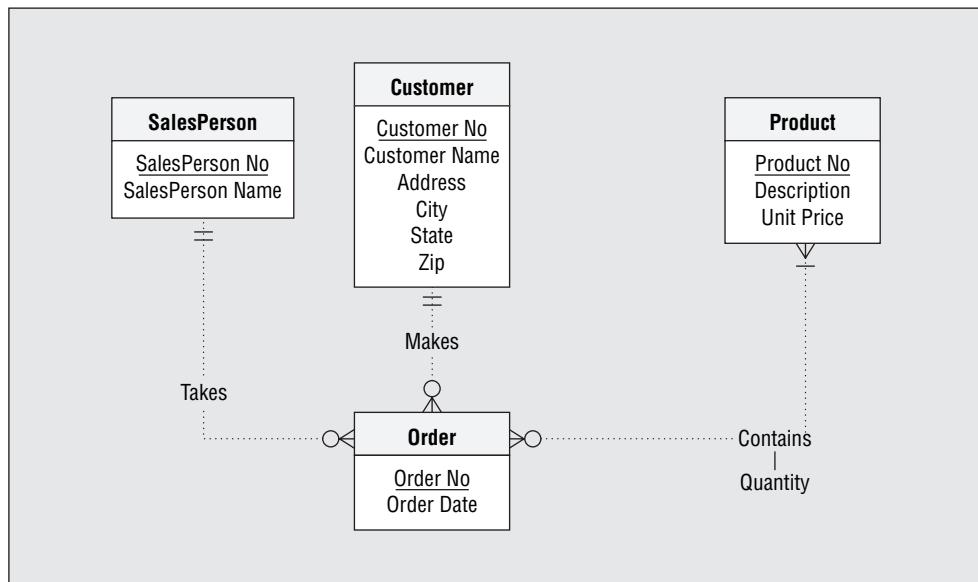
A continuación, deberá conectar los tipos de entidad derivados de los campos en el subformulario. *Product* y *OrderLine* pueden estar conectados mediante una relación 1-M. Una línea de pedido contiene un producto, pero el mismo producto puede aparecer en las líneas de pedidos de distintos formularios.

Para terminar las relaciones, necesita conectar un tipo de entidad derivado de los campos del formulario principal con un tipo de entidad derivado de los campos de los subformularios.

**FIGURA 12.7**  
Diagrama de entidad-relación para el formulario de pedido de cliente



**FIGURA 12.8**  
ERD alternativo para el formulario de pedido del cliente



Por lo regular, la relación conecta el tipo de entidad del formulario (*Order*) con un tipo de entidad derivado del nodo hija. Esta relación puede ser 1-M o M-N. En la figura 12.7 podemos asumir que un pedido suele estar relacionado con muchos productos. Si analiza otros ejemplos de formulario de pedido, podrá ver el mismo producto asociado con distintos pedidos. Por tanto, un producto puede estar relacionado con muchos pedidos. Aquí es importante señalar que *Quantity* no está asociado con *Product* ni *Order*, pero sí con la combinación. Esta última llega a considerar una relación o tipo de entidad. En la figura 12.7, *OrderLine* es un tipo de entidad. La figura 12.8 muestra una representación alternativa como una relación M-N.

#### Paso 5: Revisar completez y consistencia

En el quinto paso tiene que revisar el ERD para verificar que esté completo y que haya consistencia con la estructura del formulario. El ERD se debe apegar a las reglas del diagrama definidas en el capítulo 5 (sección 5.4.2). Por ejemplo, el ERD debe contener cardinalidades mínimas

y máximas para todas las relaciones, una llave primaria para todos los tipos de entidad y un nombre para todas las relaciones.

Para que haya consistencia, la estructura del formulario ofrece varias limitaciones en las cardinalidades de la relación, como lo resume la tabla 12.2. La primera regla es necesaria porque sólo se muestra un valor en el formulario. Por ejemplo, hay un solo valor para el número de cliente, el nombre, etc. Como ejemplo de la primera regla, la cardinalidad máxima es una en la relación de *Order* con *Customer* y de *Order* con *Salesperson*. La segunda regla garantiza que haya una relación 1-M del nodo madre al hija. Un registro determinado en el nodo madre puede estar relacionado con varios registros en el nodo hija. Como ejemplo de la segunda regla, la relación de *Order* con *OrderLine* tiene una cardinalidad máxima de M. En el ERD alternativo (figura 12.8), la cardinalidad máxima de *Order* con *Product* es M.

Después de seguir los pasos del análisis de formularios, también podrá explorar las transformaciones que se estudian en el capítulo 6 (sección 6.2). Con frecuencia es útil el atributo para la transformación del tipo de entidad. Si el formulario sólo muestra una llave primaria, quizás no cree un tipo de entidad en principio. Por ejemplo, si sólo aparece el número de vendedor, tal vez no cree un tipo de entidad independiente para el vendedor. Puede preguntar al usuario si es preciso conservar otros datos acerca del vendedor. Si es así, transforme el número de vendedor en un tipo de entidad.

#### Otro ejemplo de análisis de formulario

El formulario de factura (figura 12.9) ofrece otro ejemplo de análisis de formularios. Un cliente recibe un formulario de factura con los productos que pidió. En el formulario principal, una factura contiene campos para identificar el cliente y el pedido. En el subformulario, una factura identifica los productos y las cantidades enviadas, solicitadas y pendientes. La cantidad

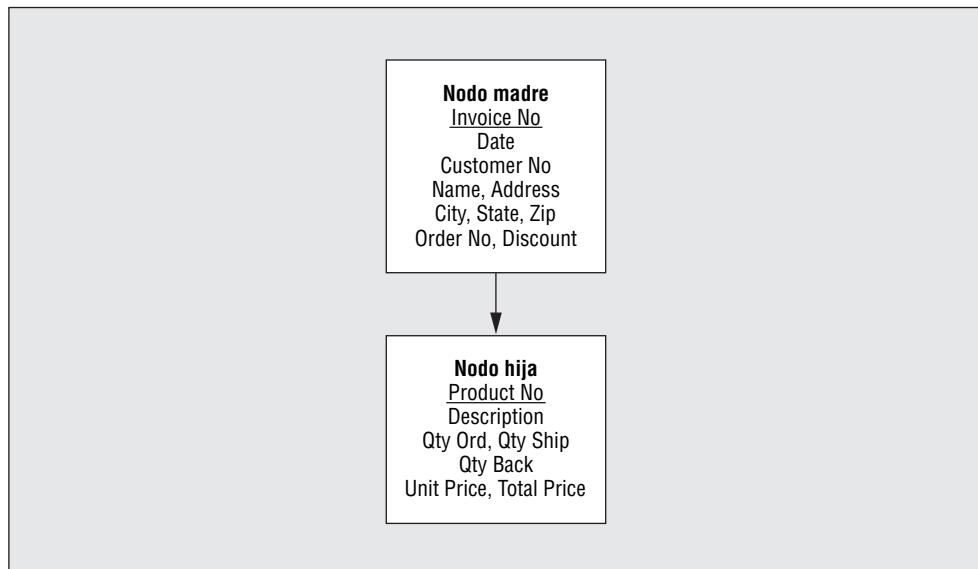
**TABLA 12.2**  
Reglas de consistencia  
para las cardinalida-  
des de las relaciones

1. La cardinalidad máxima debe ser una, por lo menos en una dirección, para las relaciones que conectan los tipos de entidad derivados del mismo nodo (madre o hija).
2. La cardinalidad máxima debe ser mayor que uno, por lo menos en una dirección, para las relaciones que conectan los tipos de entidad derivados de nodos en distintos niveles de la jerarquía de formularios.

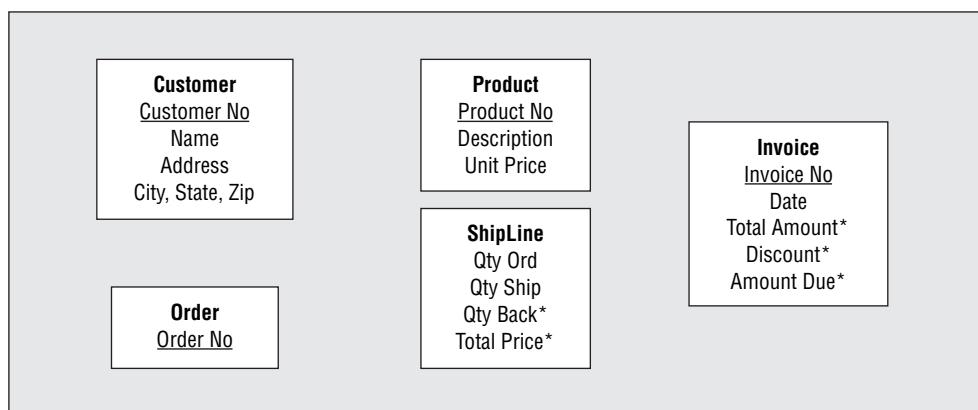
**FIGURA 12.9**  
Ejemplo de forma-  
rio de factura

Invoice Form						
<b>Customer No.:</b> 1273 <b>Name:</b> Contemporary Designs <b>Address:</b> 123 Any Street <b>City:</b> Seattle				<b>Invoice No.:</b> 06389 <b>Date:</b> 3/28/2006 <b>Order No.:</b> 61384 <b>Zip:</b> 98105		
Product No.	Description	Qty. Ord.	Qty. Ship.	Qty. Back.	Unit Price	Total Price
B381	Cabinet	2	2		150.00	300.00
R210	Table	1	1		500.00	500.00
M128	Bookcase	4	2	2	200.00	400.00
						Total Amount \$1200.00
						Discount 60.00
						Amount Due \$1140.00

**FIGURA 12.10**  
Estructura jerárquica para el formulario de factura



**FIGURA 12.11**  
Tipos de entidad para el formulario de factura



pendiente es igual a la cantidad pedida menos la cantidad enviada. La figura 12.10 muestra la estructura jerárquica relacionada con este formulario.

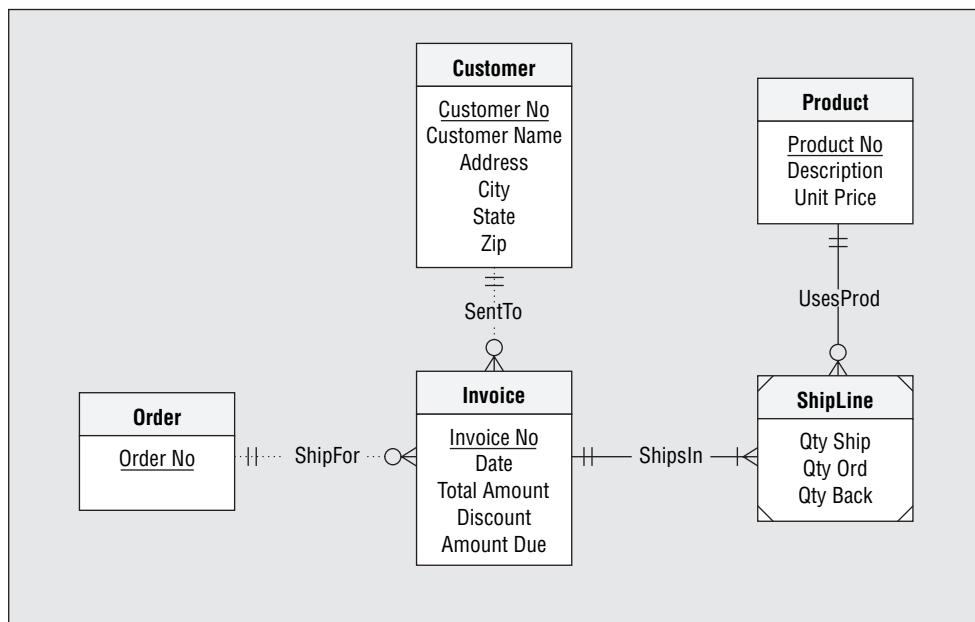
La figura 12.11 muestra el resultado de los pasos 2 y 3 para el formulario factura de cliente. Los asteriscos indican los campos calculados. *Invoice*, *Customer* y *Order* se derivan del nodo madre. *Product* y *ShipLine* se derivan del nodo hija. Si le hace falta *ShipLine* puede agregarlo más adelante como relación.

La figura 12.12 muestra el ERD para el formulario de factura. Las relaciones *SentTo* y *ShipFor* conectan los tipos de entidad del nodo madre. La relación *ShipsIn* conecta un tipo de entidad en el nodo madre (*Invoice*) con un tipo de entidad en el nodo hija (*ShipLine*). La figura 12.13 muestra un ERD alternativo con el tipo de entidad *ShipLine* reemplazado con una relación M-N.

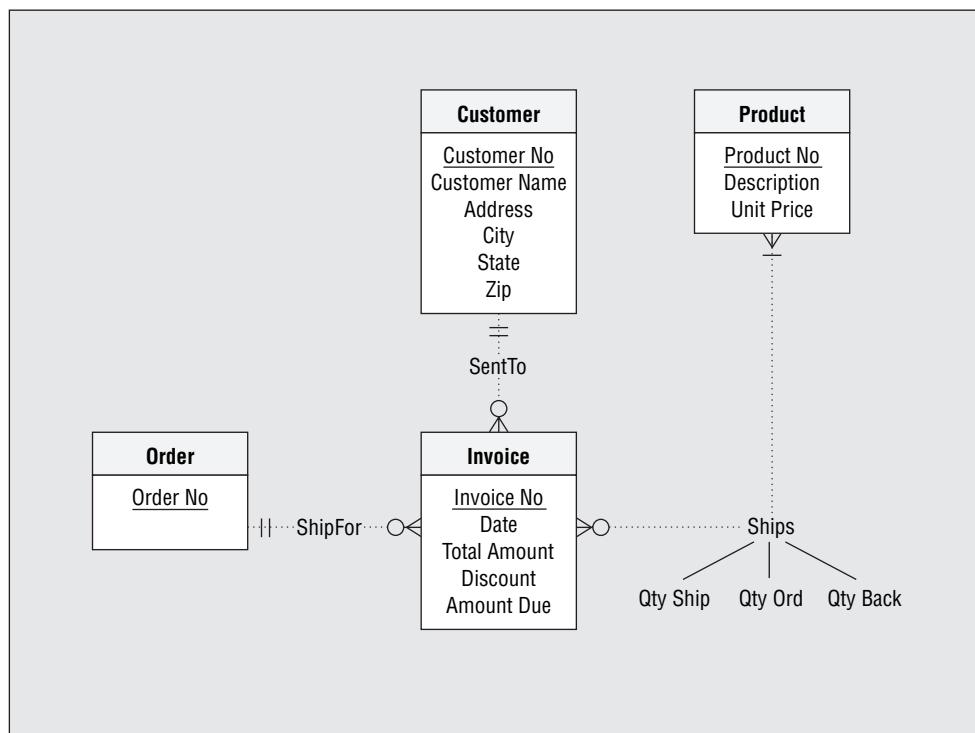
### 12.2.2 Análisis de las relaciones M-way utilizando formularios

El capítulo 7 describió el concepto de independencia de relaciones como una forma de razonar sobre la necesidad de relaciones M-way. Esta sección describe una manera de razonar más orientada a aplicaciones con respecto a las relaciones M-way. Puede usar formularios de captura de datos para ayudar a determinar si es necesario un tipo de entidad asociativo para representar una relación M-way que comprende tres o más tipos de entidad. Los formularios de captura de datos proporcionan un contexto para entender las relaciones M-way. Sin el contexto de un

**FIGURA 12.12**  
ERD para el formulario de factura



**FIGURA 12.13**  
ERD alternativo para el formulario de factura



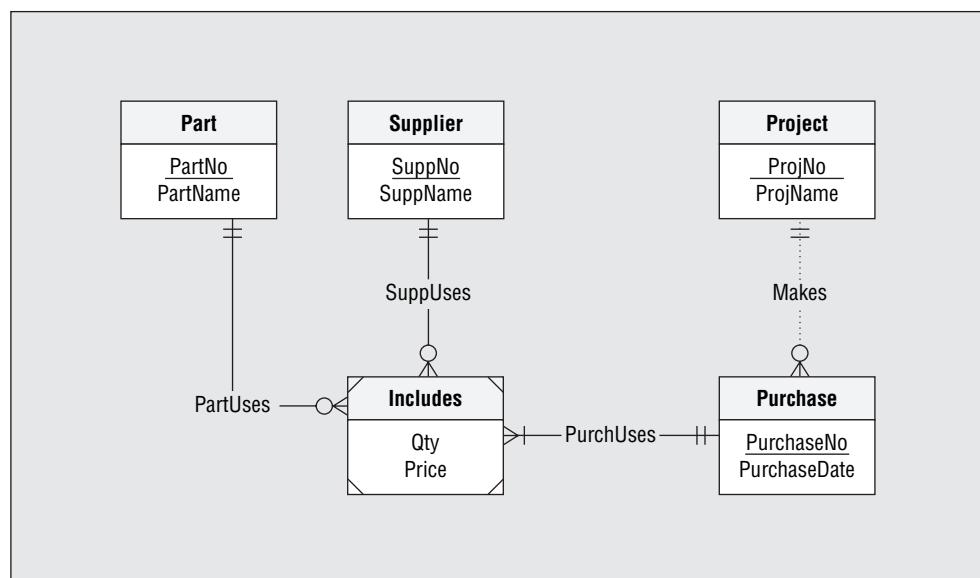
formulario, puede ser difícil determinar la necesidad de una relación M-way en lugar de las relaciones binarias.

Una relación M-way puede ser necesaria si un formulario muestra un patrón de captura de datos que comprende tres tipos de entidad. Por lo general, un tipo de entidad reside en el formulario principal y los otros dos en el subformulario. La figura 12.14 muestra un formulario con un proyecto en el formulario principal y combinaciones de partes-proveedores (dos tipos de entidad) en el subformulario. Este formulario puede usarse para comprar partes para un proyecto

**FIGURA 12.14**  
Ejemplo de formulario de compra para un proyecto

Project Purchasing Form			
Purchase No.: P1234		Purchase Date: 3/19/2006	
Project No.: PR1		Project Manager: Jon Smith	
Part No.	Supplier No.	Quantity	Unit Price
M128	S100	4	\$120
M128	S101	3	\$150
R210	S102	1	\$500

**FIGURA 12.15**  
Diagrama de entidad-relación para el formulario de compra de un proyecto



**FIGURA 12.16**  
Ejemplo de formulario de compra

Purchasing Form			
Purchase No.: P1234		Purchase Date: 3/19/2006	
Supplier No.: S101		Supplier Name: Anytime Supply	
Part No.	Quantity		Unit Price
M128	4		\$120
M129	3		\$150
R210	1		\$500

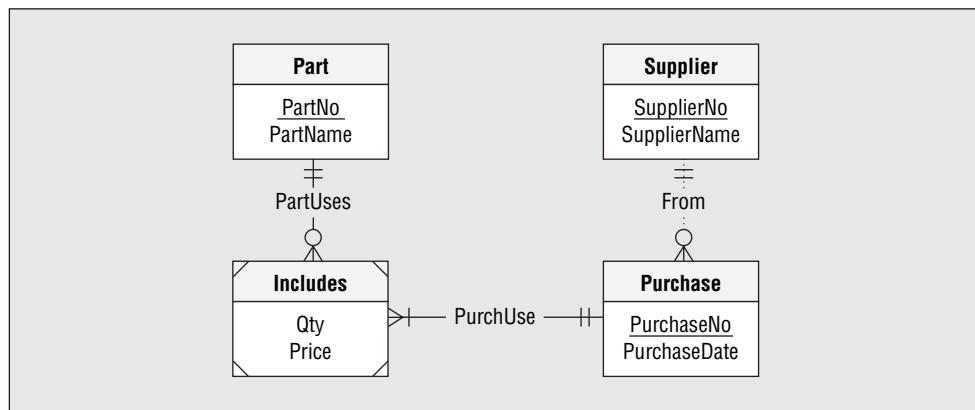
en particular (compras localizadas). Antes de tomar las decisiones de compra en los proyectos es posible actualizar tanto el número de parte como el número de proveedor en el subformulario. La figura 12.15 muestra un ERD para este formulario. Es necesario contar con un tipo de entidad asociativa que comprenda compra, parte y proveedor en el formulario principal, y partes relacionadas (un tipo de entidad) en el subformulario.

Como una alternativa para cada proyecto de compra, algunas empresas pueden centralizar el proceso de compras. La figura 12.16 muestra un formulario de compra con el soporte centralizado en el proveedor y una relación de partes (una por cada tipo) en la subforma.

El ERD de la figura 12.17 muestra una relación binaria entre *Purchase* y *Part*. Para asignar las partes en los proyectos, hay otro formulario con el proyecto en el formulario principal y las partes utilizadas por el proyecto en el subformulario. El ERD para el otro formulario necesitaría una relación binaria entre el proyecto y la parte.

**FIGURA 12.17**

Diagrama de relación de entidades para el formulario de compra

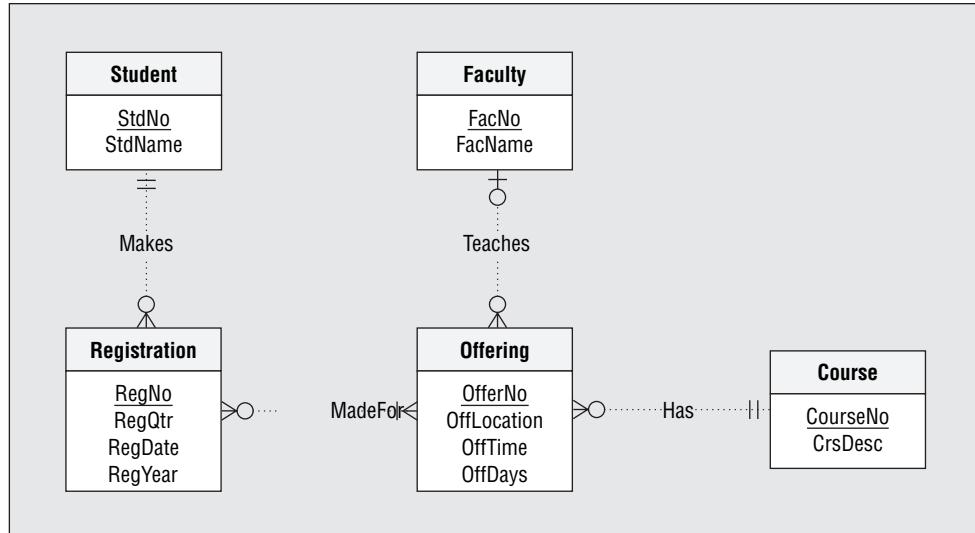
**FIGURA 12.18**

Formulario de registro

Registration No.: 1273	Date: 5/15/2006					
Quarter: Fall	Year: 2006					
Student No.: 123489	Student Name: Sue Thomas					
<hr/>						
Offer No.	Course No.	Days	Time	Location	Faculty No.	Faculty Name
1234	IS480	MW	10:30	BLM211	1111	Sally Hope
3331	IS460	MW	8:30	BLM411	2121	George Jetstone
2222	IS470	MW	1:30	BLM305	1111	Sally Hope

**FIGURA 12.19**

Diagrama de entidad-relación para el formulario de registro



Aun cuando haya dos o más tipos de entidad en un subformulario, podrían ser suficientes las relaciones binarias si sólo se puede actualizar un tipo de entidad. En la figura 12.14, tanto Supplier No. como Part No. se pueden actualizar en el subformulario. Por ello es necesaria una relación M-way. Como ejemplo contrario, la figura 12.18 muestra un formulario para el registro de cursos. El subformulario muestra llaves primarias de los tipos de entidad *Offering*, *Faculty* y *Course*, pero sólo *Offering* se puede actualizar en el subformulario. Faculty No. y Course No. son de sólo-lectura. La selección de un profesor y el curso correspondiente se hace en otros formularios. Por tanto, el ERD sólo contiene relaciones binarias, como muestra la figura 12.19.

## 12.3 Integración de vistas

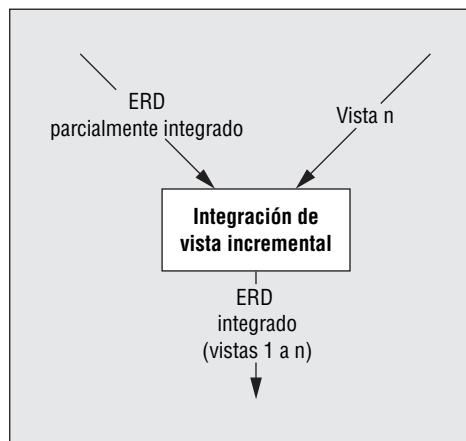
En un gran proyecto de base de datos, incluso los diseñadores de base de datos mejor capacitados necesitan herramientas para manejar la complejidad del proceso de diseño. El diseño y la integración de vistas en conjunto le ayudan a manejar un gran proyecto de diseño de bases de datos permitiéndole dividir una tarea extensa en partes menores. En la sección anterior estudió un método para diseñar un ERD que represente los requisitos de datos de un formulario. Esta sección describe el proceso para combinar vistas individuales en el diseño de una base de datos completa. Presentamos dos planteamientos para la integración de vistas, además de un ejemplo de cada uno.

### 12.3.1 Enfoque de integración incremental y paralelo

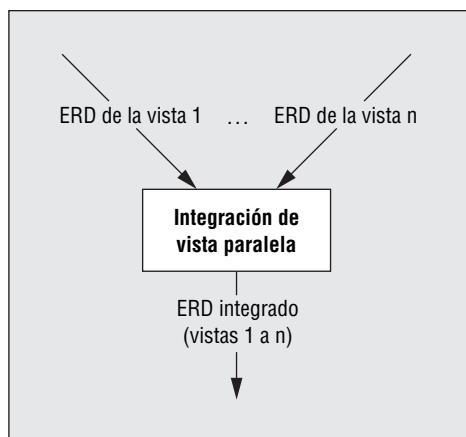
Los enfoques incremental y paralelo son formas opuestas de llevar a cabo la integración de vistas. En el enfoque incremental (figura 12.20) se combinan una vista y un ERD parcialmente integrado en cada paso de la integración. En un principio, el diseñador elige una vista y construye un ERD para ésta. Para las vistas posteriores, el diseñador lleva a cabo la integración al tiempo que analiza la vista siguiente. Los procesos de diseño e integración de vistas se realizan en forma conjunta para cada vista después de la primera. Este enfoque es incremental, ya que después de cada paso se produce un ERD parcialmente integrado. También es binario porque la vista actual se analiza con el ERD parcialmente integrado.

En el enfoque paralelo (figura 12.21), los ERD se producen para cada vista y luego se combinan. La integración ocurre en un paso posterior al análisis de todas las vistas. Este enfoque es

**FIGURA 12.20**  
Proceso de enfoque incremental



**FIGURA 12.21**  
Proceso de integración paralela



paralelo porque varios diseñadores pueden llevar a cabo diseños de vistas al mismo tiempo. La integración puede ser más compleja en este enfoque porque se pospone hasta que todas las vistas están terminadas. La integración ocurre en un solo paso cuando todas las vistas se integran para producir el ERD final.

Ambos planteamientos tienen ventajas y desventajas. El enfoque incremental tiene más pasos para la integración, pero cada uno de ellos es menor. El enfoque paralelo pospone la integración hasta el final, cuando quizás sea necesario un esfuerzo de integración mayor. El incremental es adecuado para vistas estrechamente relacionadas. Por ejemplo, los formularios de pedido y factura están estrechamente relacionados porque un pedido precede a una factura. El enfoque paralelo funciona bien para proyectos grandes con vistas que no están muy relacionadas. Los equipos independientes pueden trabajar en distintas partes de un diseño de forma paralela. En un proyecto grande con muchos diseñadores de bases de datos, el enfoque paralelo ofrece soporte para un trabajo más independiente.

### *Determinación de una estrategia de integración*

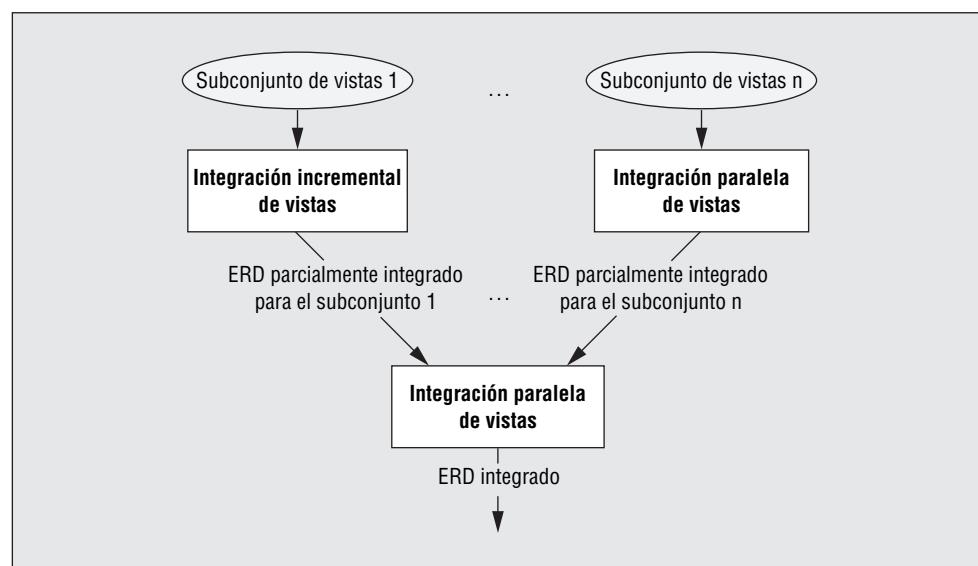
#### **estrategia de integración**

una combinación de enfoques incrementales y paralelos para integrar un conjunto de vistas. Las vistas se dividen en subconjuntos. Para cada subconjunto de vistas se utiliza la integración incremental. La integración paralela se aplica a los ERD que resultan de integrar los subconjuntos de vistas.

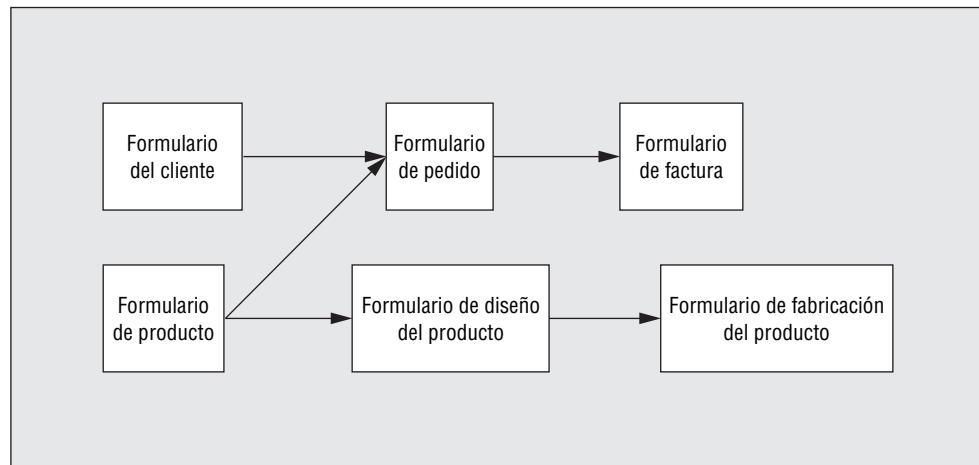
Por lo regular, los enfoques incremental y paralelo se combinan en un proyecto extenso de diseño de bases de datos. Una estrategia de integración (figura 12.22) especifica la combinación de enfoques incrementales y paralelos para integrar un conjunto de vistas. Para elegir una estrategia de integración tiene que dividir las vistas en subconjuntos (digamos  $n$  subconjuntos). El enfoque incremental se sigue para cada subconjunto de vistas. Debe elegir los subconjuntos de vistas de modo que las vistas estén estrechamente relacionadas en cada uno de ellos. Las vistas en subconjuntos diferentes no deben estar muy relacionadas. La integración incremental entre los subconjuntos de vistas puede proceder en paralelo. Después de producir un ERD integrado para cada subconjunto de vistas, una integración paralela produce el ERD integrado final. Si los ERD de cada subconjunto de vistas no se superponen en gran medida, la integración final no debe ser difícil. Si hay una superposición importante entre el subconjunto de vistas, es posible usar la integración incremental para combinar los ERD de los subconjuntos de vistas.

Por ejemplo, considere una base de datos que ofrece soporte para una empresa de consultoría. La base de datos debe apoyar las funciones de marketing para los clientes potenciales, la facturación de los proyectos existentes y el trabajo que se realiza en éstos. El esfuerzo de diseño de la base de datos se puede dividir en tres partes (marketing, facturación y trabajo). Un equipo de diseño independiente puede trabajar de manera incremental en cada parte. Si la parte de marketing tiene requisitos de contactos con clientes y promociones, es necesario producir dos ERD. Después de trabajar de manera independiente, los equipos pueden realizar una integración paralela para combinar su trabajo.

**FIGURA 12.22**  
Resumen de una estrategia de integración general



**FIGURA 12.23**  
Relaciones de precedencia entre formularios



### *Relaciones de precedencia entre formularios*

Para ayudar a determinar una estrategia de integración, deberá identificar las relaciones de precedencia entre formularios. El formulario A precede al formulario B si el formulario A se debe terminar antes de usar el formulario B. Por lo regular, el formulario A proporciona algunos datos que se utilizan en el formulario B. Por ejemplo, el formulario de factura (figura 12.9) usa la cantidad de cada producto pedido (del formulario de pedido) para determinar la cantidad a enviar. Una buena regla general es colocar los formularios con relaciones de precedencia en el mismo subconjunto. Por tanto, los formularios de factura y pedido deben estar en el mismo subconjunto de vistas.

Para ilustrar con más detalle el uso de las relaciones de precedencia, vamos a ampliar el ejemplo de los pedidos y facturas. La figura 12.23 muestra las relaciones de precedencia entre los formularios de una empresa de manufactura. El formulario de diseño de producto contiene datos acerca de los componentes del producto. El formulario de fabricación de producto contiene información sobre la secuencia de operaciones físicas necesarias para fabricar un producto. Los formularios de cliente y producto contienen datos acerca de los clientes y productos, respectivamente. Las relaciones de precedencia indican que es preciso completar el formulario de cliente y producto antes de tomar un pedido. De modo similar, es preciso completar los formularios de producto y diseño de producto antes de completar un formulario de manufactura.

Utilizando estas relaciones de precedencia, podemos dividir los formularios en dos grupos: (1) un proceso de pedidos, que consiste en formularios de cliente, producto, pedido y factura, y (2) un proceso de manufactura que consiste en formularios de producto, diseño de producto y formularios de manufactura de producto.

### *Identificación de sinónimos y homónimos*

La identificación de sinónimos y homónimos es un aspecto muy importante en cualquier planteamiento de integración. Un sinónimo es un grupo de palabras que se escriben diferente pero tienen el mismo significado. Por ejemplo, OrdNo, Order Number y ONO son sinónimos. Éstos ocurren cuando distintas partes de una organización usan un vocabulario diferente para describir los mismos conceptos. Esta situación es probable sobre todo si antes del esfuerzo de diseño no existía una base de datos común.

Un homónimo es un grupo de palabras que tienen el mismo sonido y a menudo se escriben igual, pero tienen significados diferentes. En el diseño de bases de datos, los homónimos surgen gracias al contexto de uso. Por ejemplo, dos formularios pueden mostrar un campo de dirección. En uno de ellos, la dirección puede representar la dirección de la calle, mientras en la otra aparece calle, ciudad, estado y código postal. Aun cuando ambos campos de dirección representan la dirección de la calle, no son iguales. Quizás un formulario contiene la dirección de facturación y la otra presenta la dirección de envío.

### Identificación de sinónimos y homónimos

Un sinónimo es un grupo de palabras que se escriben diferente, pero tienen el mismo significado. Un homónimo es un grupo de palabras que tienen el mismo sonido y a menudo se escriben igual, pero tienen significados diferentes. El uso de estándares para nombres y un diccionario de datos corporativos ayuda en la identificación de sinónimos y homónimos.

La estandarización del vocabulario es parte importante del desarrollo de bases de datos. Para estandarizarlo, es preciso identificar los sinónimos y homónimos. El uso de estándares para nombres y un diccionario de datos corporativos ayudan en la identificación de sinónimos y homónimos. Puede crear y mantener un diccionario de datos corporativos con una herramienta CASE. Algunas herramientas CASE ayudan a implementar los estándares de nombres. Pero aun con estas herramientas puede ser difícil reconocer los sinónimos y los homónimos. El punto más importante es estar alerta de su existencia. Solucionarlos es más fácil: dé el mismo nombre a los sinónimos (o establezca una lista oficial de sinónimos) y dé un nombre diferente a cada homónimo.

### 12.3.2 Ejemplos de integración de vistas

Esta sección ilustra los enfoques incremental y paralelo para la integración de vistas, utilizando los formularios de pedido y factura. El resultado final es idéntico con ambos enfoques, pero el camino para llegar a éste es diferente.

#### Ejemplo de integración incremental

Para demostrar el enfoque de integración incremental vamos a integrar el formulario de factura (figura 12.9) con el ERD de la figura 12.7. La estructura jerárquica del formulario de factura se muestra en la figura 12.10. Puede empezar por agregar una entidad para la factura con el número y la fecha de ésta. Al llevar a cabo los pasos 2 y 3 (figura 12.11), resulta útil ver cómo se deben combinar los tipos de entidades en el ERD existente (figura 12.7). A continuación presentamos los otros campos del formulario que coinciden con los tipos de entidades existentes.

- Order No. coincide con el tipo de entidad *Order*.
- Customer No., Customer Name, Address, City, State y Zip coinciden con el tipo de entidad *Customer*.
- Product No., Description y Unit Price coinciden con el tipo de entidad *Product*.

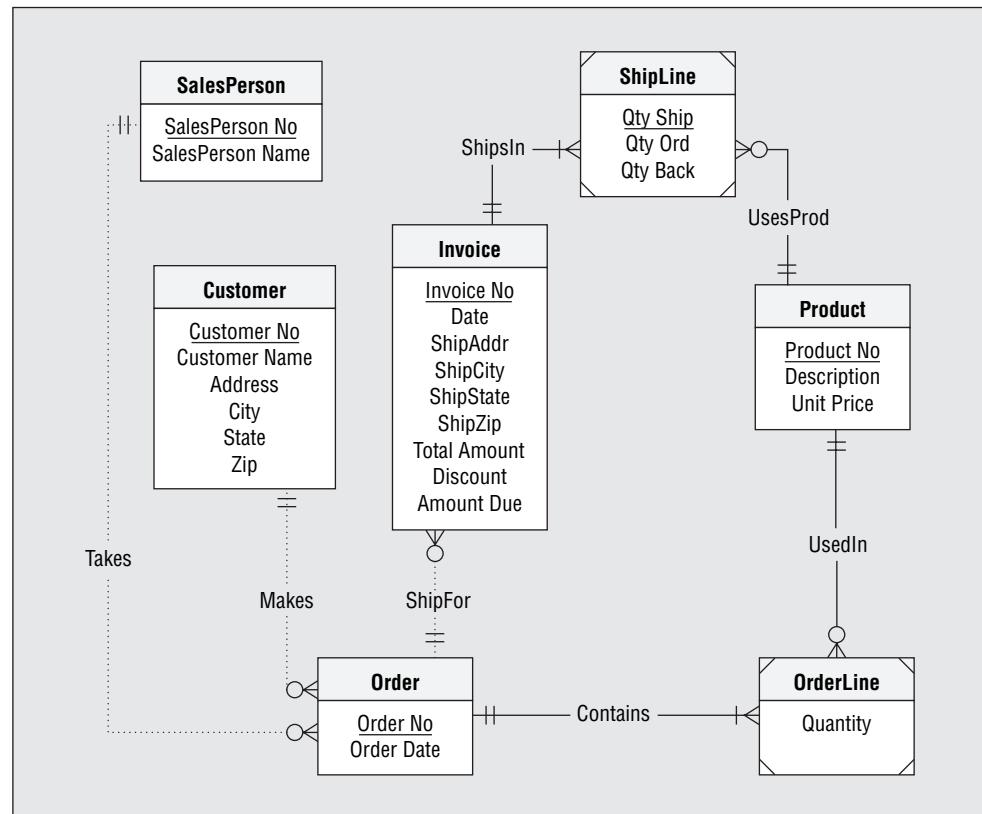
Al comparar los campos del formulario con los tipos de entidad existentes, debe revisar los sinónimos y los homónimos. Por ejemplo, no queda claro si *Address*, *City*, *State* y *Zip* tienen el mismo significado en ambos formularios. Desde luego, estos campos tienen el mismo significado general; sin embargo, no es evidente si un cliente tiene una dirección diferente para el pedido y el envío. Tal vez sea necesario realizar entrevistas adicionales y analizar otros formularios para resolver esta duda. Si determina que los dos conjuntos de campos son homónimos (un pedido se puede facturar a una dirección y enviar a otra), hay muchas alternativas para modelar los datos, como se presenta a continuación.

- Revisar el tipo de entidad *Customer* con dos conjuntos de campos de dirección: campos de dirección de facturación y campos de dirección de envío. Esta solución limita al cliente a una sola dirección de envío. Si existe la posibilidad de que el cliente tenga más de una dirección de envío, esta solución no es factible.
- Agregar los campos de dirección de envío al tipo de entidad *Invoice*. Esta solución ofrece soporte para varias direcciones de envío por cliente. No obstante, si se borra una factura, la dirección de envío se pierde.
- Crear un nuevo tipo de entidad (*ShipAddress*) con los campos de la dirección de envío. Esta solución ofrece soporte para varias direcciones de envío por cliente. Tal vez requiera de un esfuerzo adicional para recopilar las direcciones de envío. Esta solución es la mejor si las direcciones de envío se mantienen separadas de las facturas.

El ERD integrado en la figura 12.24 utiliza la segunda alternativa. En un problema real es necesario recopilar más información de los usuarios antes de tomar la decisión.

En el proceso de integración incremental es necesario seguir el proceso usual de conectar los tipos de entidad (paso 4 en la figura 12.3). Por ejemplo, hay una cardinalidad M que relaciona el tipo de entidad derivado del nodo madre con un tipo de entidad derivado del nodo hija. La cardinalidad máxima en *ShipsIn* de *Invoice* a *ShipLine* satisface esta limitación. Cabe señalar que *ShipLine* podría representarse como una relación M-N, en lugar de un tipo de entidad con dos relaciones 1-M.

**FIGURA 12.24**  
Diagrama de relación de entidades integradas



Otro punto interesante de la figura 12.24 es que no existe una relación de *Invoice* con *Customer*. En principio podría parecer que es necesaria una relación porque los datos del cliente aparecen en el formulario principal de una factura. Si el cliente en la factura puede ser diferente del cliente del pedido, necesitamos una relación entre *Invoice* y *Customer*. Si el cliente en el pedido es el mismo que en la factura relacionada, esta relación no es necesaria. El cliente para una factura se puede encontrar navegando de *Invoice* a *Order* y de *Order* a *Customer*. En la figura 12.24 se supone que el cliente del pedido y el cliente de la factura son idénticos.

#### Ejemplo de integración paralela

Para demostrar el proceso de integración paralela, vamos a integrar el formulario de factura (figura 12.9) con el formulario de pedido (figura 12.2). La principal diferencia entre los planteamientos paralelo e incremental es que, en el primero, la integración ocurre de manera posterior. Por tanto, el primer paso consiste en construir un ERD para cada formulario siguiendo los pasos para el análisis de formulario que describimos anteriormente. En el ERD para el formulario de factura (figura 12.12), *Invoice* está directamente conectada con *Customer* y *Order*. La conexión directa sigue la práctica de convertir el tipo de entidad del formulario (*Invoice*) en el centro del diagrama.

El proceso de integración combina el ERD del formulario de pedido (figura 12.7) con el ERD del formulario de factura (figura 12.12) para producir el ERD que muestra la figura 12.24. El ERD final debe ser el mismo, sin importar si emplea el planteamiento incremental o paralelo.

Una vez más, un aspecto importante de la integración es la solución de los homónimos para los campos de dirección. En los dos ERD (figuras 12.7 y 12.12), el tipo de entidad *Customer* contiene los campos de dirección. Al trabajar de manera independiente en ambos formularios, es fácil pasar por alto los dos usos de los campos de dirección: facturación y envío. A menos que observe que los campos de dirección en el formulario de factura son para propósitos de envío, tal vez no se dé cuenta de que son homónimos.

Otro aspecto de la integración son las conexiones entre *Invoice*, *Order* y *Customer*. En la figura 12.7, *Customer* está directamente conectado con *Order*, pero en la figura 12.12, *Order* y *Customer* no están conectados de manera directa mediante una relación. El proceso de integración debe solucionar esta diferencia. La relación entre *Order* y *Customer* es necesaria porque los pedidos preceden a las facturas. Una relación entre *Invoice* y *Customer* no es necesaria si el cliente que aparece en una factura es el mismo que aparece en el pedido relacionado. Suponiendo que el cliente en un pedido es idéntico al que aparece en las facturas relacionadas, *Invoice* no tiene una conexión directa con *Customer* en la figura 12.24.

Estos dos ejemplos de integración ilustran la ventaja del planteamiento de integración incremental sobre el paralelo. Los conflictos debidos a los distintos usos de campos y tiempo (los pedidos preceden a las facturas) se resuelven rápidamente en el planteamiento incremental. En el paralelo, estos conflictos no se detectan sino hasta el último paso. Este estudio nos lleva a la conclusión a la que habíamos llegado anteriormente: la integración incremental es la más apropiada al integrar vistas con una relación estrecha.

## Reflexión final

Este capítulo describió el diseño y la integración de vistas, una habilidad importante para diseñar bases de datos extensas. Estas bases pueden comprender ERD con cientos de tipos de entidad y relación. Además del tamaño extenso de los ERD, a menudo hay cientos de formas, reportes, procedimientos almacenados y disparadores que utilizan la base de datos. El diseño y la integración de vistas ayudan a manejar la complejidad de estos esfuerzos de diseño de grandes bases de datos.

Este capítulo enfatizó el uso de los formularios en el proceso de diseño de vistas. Los formularios son una fuente de requerimientos importante porque son comunes y se comunican con facilidad. Se presentó un procedimiento de cinco pasos para analizar un formulario. El resultado del proceso de análisis de formularios es un ERD que captura los requerimientos de los datos del formulario. Este capítulo también describió de qué manera el proceso de análisis de formularios ayuda a detectar la necesidad del tipo de entidad asociativa M-way en un ERD.

Este capítulo describió dos planteamientos para la integración de vistas. En el planteamiento incremental, una vista y el ERD parcialmente integrado se combinan en cada paso de la integración. En el planteamiento paralelo, se producen ERD para cada vista y luego se combinan. El planteamiento incremental funciona bien para las vistas no relacionadas. Este capítulo estudió cómo determinar una estrategia de integración para combinar los planteamientos incremental y paralelo. En cualquier planteamiento, es crítico solucionar los sinónimos y los homónimos. Este capítulo demostró que los formularios proporcionan un contexto para solucionar los sinónimos y homónimos.

En este capítulo finaliza el estudio de las dos primeras etapas del desarrollo de bases de datos: modelado conceptual de datos y diseño lógico de bases de datos, y ofrece un vínculo entre el desarrollo de aplicaciones y de bases de datos. Al terminar este proceso, usted deberá tener un diseño de bases de datos relacional de alta calidad: un diseño que representa las necesidades de la organización y que está libre de redundancias no deseadas. El capítulo 13 ofrece un estudio de caso detallado para aplicar las ideas de las partes 2 a 5 de este libro.

## Revisión de conceptos

- Medidas de complejidad organizacional y bases de datos.
- Características de los esfuerzos de diseño de bases de datos extensas.
- Insumos y productos para el diseño y la integración de vistas.
- Importancia de los formularios como fuentes de requisitos para las bases de datos.
- Cinco pasos del análisis de formularios.
- Estructura del formulario: nodos y llaves de nodo.
- Reglas para agregar las relaciones en el análisis de formularios.
- Reglas para revisar las cardinalidades para la consistencia en el análisis de formularios.

- Uso del análisis de formularios para detectar la necesidad de relaciones M-way.
- Uso de los planteamientos de integración incremental y paralela.
- Estrategia de integración: una combinación de los planteamientos incremental y paralelo para integrar un conjunto de vistas.
- Uso de las relaciones de precedencia entre formularios para determinar una estrategia de integración.
- Detección de sinónimos y homónimos durante la integración de vistas.

## Preguntas

1. ¿Qué factores influyen en el tamaño de un esquema conceptual?
2. ¿Cuáles son las medidas de complejidad en el diseño de bases de datos conceptuales?
3. ¿De qué manera el proceso de diseño e integración de vistas ayuda a manejar la complejidad de los esfuerzos de diseño de bases de datos extensas?
4. ¿Cuál es el objetivo del análisis de formularios?
5. ¿Qué nivel de detalle se debe proporcionar para las definiciones de los formularios con el fin de apoyar el proceso de análisis de formularios?
6. ¿Qué son las llaves de nodo en la estructura de un formulario?
7. ¿De qué manera los nodos en la estructura del formulario corresponden con los formularios principales y subformularios?
8. ¿Qué es el tipo de entidad de un formulario?
9. ¿Por qué el ERD para un formulario a menudo tiene una estructura diferente al del formulario?
10. ¿Por qué es recomendable colocar el tipo de entidad del formulario en el centro del ERD?
11. Explique la primera regla de consistencia en la tabla 12.2.
12. Explique la segunda regla de consistencia en la tabla 12.2.
13. ¿Qué patrón en un formulario de captura de datos puede indicar la necesidad de una relación M-way?
14. ¿Cuántos pasos de integración son necesarios para realizar una integración incremental con 10 vistas?
15. ¿Cuántos pasos de diseño de vistas son necesarios para realizar una integración paralela con 10 vistas?
16. En el planteamiento de integración incremental, ¿por qué se llevan a cabo el diseño y la integración de vistas al mismo tiempo?
17. ¿Cuándo es apropiado el planteamiento de integración incremental?
18. ¿Cuándo es apropiado el planteamiento de integración paralelo?
19. ¿Qué es una estrategia de integración?
20. ¿En qué momento un formulario depende de otro?
21. ¿Qué criterios pueden emplearse para decidir cómo agrupar las vistas en una estrategia de integración?
22. ¿Qué es un sinónimo en una integración de vistas?
23. ¿Qué es un homónimo en una integración de vistas?
24. ¿Por qué ocurren sinónimos y homónimos al diseñar una base de datos?
25. ¿De qué manera el uso de formularios en el diseño de bases de datos le ayuda a detectar sinónimos y homónimos?

## Problemas

Además de los problemas aquí presentados, el sitio web de este libro ofrece una práctica adicional de estudios de caso. Para complementar los ejemplos de este capítulo, el capítulo 13 proporciona un caso completo de diseño de base de datos que incluye diseño e integración de vistas.

1. Realice un análisis de formulario para el formulario simple de pedido (problema 22 del capítulo 10). Su solución debe incluir una estructura jerárquica para el formulario, un ERD que represente el formulario y justificaciones de diseño. Ignore el diseño de bases de datos del capítulo 10 al realizar el análisis. En este análisis puede suponer que un pedido debe contener por lo menos un producto.
2. Realice un análisis de formularios para el formulario de pedido (problema 23 del capítulo 10). Su solución debe incluir una estructura jerárquica para el formulario, un ERD que represente el formulario y justificaciones de diseño. Ignore el diseño de bases de datos del capítulo 10 al realizar el análisis. A

A continuación se presentan varios puntos adicionales para complementar el formulario de muestra que aparece en el problema 23 del capítulo 10:

- En todos los formularios adicionales, los datos del cliente aparecen en el formulario principal.
  - En algunos formularios adicionales, los datos del empleado no aparecen en el formulario principal.
  - En algunos formularios adicionales, el precio para el mismo producto varía. Por ejemplo, el precio para el producto P0036566 es de 169 dólares en el formulario adicional y de 150 dólares en otros formularios.
  - En algunos formularios, el precio para el mismo producto varía. Además, el número y nombre de proveedor son idénticos en todos los casos del subformulario con el mismo número de producto.
3. Realice un análisis de formularios para el formulario simple de compras (problema 25 del capítulo 10). Su solución debe incluir una estructura jerárquica para el formulario, un ERD que represente el formulario y justificaciones de diseño. Ignore el diseño de bases de datos del capítulo 10 al realizar el análisis. A continuación se presentan varios puntos adicionales para complementar el formulario de muestra que aparece en el problema 25 del capítulo 10:
- El precio unitario de la compra puede variar entre los formularios que contienen el mismo número de producto.
  - Una compra debe contener por lo menos un producto.
4. Realice un análisis de formularios para el formulario de compras (problema 26 del capítulo 10). Su solución debe incluir una estructura jerárquica para el formulario, un ERD que represente el formulario y justificaciones de diseño. Ignore el diseño de bases de datos del capítulo 10 al realizar el análisis. A continuación se presentan algunos puntos adicionales para complementar el formulario de muestra que aparece en el problema 26 del capítulo 10:
- En todos los formularios adicionales, los datos del proveedor aparecen en el formulario principal.
  - El precio de venta puede variar entre los formularios que contienen el mismo número de producto.
  - El costo unitario y el QOH son idénticos en todos los subformularios para un producto determinado.
5. Realice un análisis de formularios para el formulario de proveedores (problema 27 del capítulo 10). Su solución debe incluir una estructura jerárquica para el formulario, un ERD que represente el formulario y justificaciones de diseño. Ignore el diseño de bases de datos del capítulo 10 al realizar el análisis. En su análisis puede suponer que un producto determinado sólo aparece en un formulario de proveedor.
6. Lleve a cabo una integración paralela utilizando los ERD que creó en los problemas 2, 4 y 5. Ignore el diseño de bases de datos del capítulo 10 al realizar el análisis. Al llevar a cabo la integración, debe suponer que cada producto en un formulario de compra proviene del mismo proveedor. Además, debe suponer que es necesario completar un formulario de proveedor antes de pedir y comprar los productos.
7. Realice un análisis de formularios para el formulario de personal de proyecto en la parte inferior de esta página. Los proyectos tienen un gerente, una fecha de inicio, una fecha de terminación, una categoría, un presupuesto (horas y dólares) y una lista de personal asignado. Para cada personal asignado se muestran las horas disponibles y las horas asignadas.
8. Lleve a cabo una integración incremental utilizando el ERD del problema y el siguiente formulario de programa. Un proyecto se divide en varios programas. Cada programa se asigna a un empleado. A un empleado sólo se le puede asignar un programa si está asignado al proyecto.

Formulario de personal de proyecto				
Project ID: PR1234		Project Name: A/P testing		
Category: Auditing		Manager: Scott Jones		
Budget Hours: 170		Budget Dollars: \$10,000		
Begin Date: 6/1/2006		End Date: 6/30/2006		
Staff ID	Staff Name	Avail. Hours	Assigned Hours	
S128	Rob Scott	10	10	
S129	Sharon Store	20	5	
S130	Sue Kendall	20	15	

<b>Formulario de programa</b> <b>Staff ID:</b> S128 <b>Name:</b> Rob Scott <b>Project ID:</b> PR1234 <b>Project Manager:</b> Scott Jones			
<b>Program ID</b>	<b>Hours</b>	<b>Status</b>	<b>Due Date</b>
PR1234-1	10	completed	6/25/2006
PR1234-2	10	pending	6/27/2006
PR1234-3	20	pending	6/15/2006

9. Lleve a cabo una integración incremental utilizando el ERD del problema 8 y el siguiente formulario de horarios. Este formulario permite que un empleado registre las horas trabajadas en varios programas durante el mismo periodo.

<b>Formulario de horarios</b> <b>Timesheet ID:</b> TS100 <b>Time Period No.:</b> 5 <b>Total Hours:</b> 18 <b>Staff ID:</b> S128 <b>Name:</b> Rob Scott <b>Begin Date:</b> 5/1/2006 <b>End Date:</b> 5/31/2006			
<b>Program ID</b>	<b>Hours</b>	<b>Pay Type</b>	<b>Date</b>
PR1234-1	4	regular	5/2/2006
PR1234-1	6	overtime	5/2/2006
PR1234-2	8	regular	5/3/2006

10. Defina una estrategia de integración para los formularios personal de proyecto, programa y horarios. Justifique en forma breve su estrategia de integración.

## Referencias para ampliar su estudio

El diseño y la integración de vistas se cubren con mayor detalle en libros especializados sobre diseño de bases de datos. La mejor referencia sobre diseño e integración de vistas es Batini, Ceri y Navathe (1992). Otros libros sobre diseño de bases de datos que también cubren el diseño y la integración son Nijssen y Halpin (1989) y Teorey (1999). Puede encontrar más detalles sobre la metodología para el análisis de formularios y la integración de vistas en Choobineh, Mannino, Konsynski y Nunamaker (1988) y Choobineh, Mannino y Tseng (1992). Batra (1997) ofrece una actualización reciente a este trabajo sobre el análisis de formularios.



# Capítulo 13

---

## Desarrollo de base de datos para Student Loan Limited

### Objetivos de aprendizaje

Este capítulo aplica los conocimientos y habilidades presentados en los capítulos de las partes 2 a 5 a un caso de tamaño moderado. Al finalizar este capítulo, los estudiantes habrán adquirido los siguientes conocimientos y habilidades:

- Realizar el modelado conceptual de datos para un caso comparable.
- Refinar un ERD utilizando la conversión y la normalización para un caso comparable.
- Estimar una carga de trabajo en el diseño de una tabla de tamaño moderado.
- Realizar la selección de índices para un caso comparable.
- Especificar los requerimientos de datos para las aplicaciones en un caso comparable.

### Panorama general

---

Los capítulos de las partes 2 a 5 proporcionan los conocimientos y las técnicas sobre el proceso de desarrollo de base de datos y el desarrollo de aplicaciones de bases de datos. Para el primero, usted aprendió el uso del modelo de entidad-relación (capítulos 5 y 6), refinando un esquema conceptual a través de la conversión y la normalización (capítulos 6 y 7), los procesos de modelado e integración de vistas para esfuerzos de modelado conceptual de datos (capítulo 12) y la búsqueda de una implementación eficiente (capítulo 8). Además, aprendió sobre el amplio contexto del desarrollo de bases de datos (capítulo 2). Para el desarrollo de aplicaciones, aprendió acerca de la formulación de consultas (capítulos 3 y 9), el desarrollo de aplicaciones con vistas (capítulo 10) y los procedimientos almacenados y disparadores para personalizar las aplicaciones de bases de datos (capítulo 11).

Este capítulo aplica las técnicas de desarrollo específicas de otros capítulos a un caso de tamaño moderado. Al seguir con detenimiento este caso y su solución, reforzará sus habilidades para el diseño, se dará una mejor idea del proceso de desarrollo de bases de datos y obtendrá un modelo para el desarrollo de bases de datos en casos comparables.

Este capítulo presenta un caso derivado de las discusiones con los profesionales en sistemas de información de una importante empresa de préstamos a estudiantes. Ofrecer préstamos a estudiantes es un negocio complejo debido a los distintos tipos de préstamos, las regulaciones

gubernamentales en constante cambio y las numerosas condiciones de facturación. Para adaptar el caso a este capítulo se han omitido muchos detalles. La base de datos para el sistema de información real tiene más de 150 tablas. El caso que aquí presentamos conserva los conceptos esenciales del procesamiento de préstamos para los estudiantes, pero se puede entender en un solo capítulo. Se dará cuenta de que este caso es complejo e informativo; ¡y quizás aprenda cómo hacer para que le condonen su préstamo de estudiante!

## 13.1 Descripción del caso

---

Esta sección describe el propósito y el entorno del procesamiento de préstamos para estudiantes, así como el flujo de trabajo de un sistema propuesto para Student Loan Limited. Además de los detalles en esta sección, el apéndice 13.A contiene un glosario de los campos que contienen los formularios y reportes.

### 13.1.1 Panorama general

El programa Guaranteed Student Loan (GSL) se creó para ayudar a los estudiantes a pagar su educación universitaria. Los préstamos de GSL se clasifican de acuerdo con las condiciones del subsidio: (1) subsidiados, en los que el Gobierno Federal paga el interés acumulado durante los años de estudio, y (2) no subsidiados, en los que el Gobierno Federal no paga el interés acumulado durante los años de estudio. En los préstamos no subsidiados, el interés acumulado durante los años de estudio se suma a la cantidad principal al iniciar el pago. El pago de los préstamos empieza aproximadamente seis meses después de la separación del colegio. Un estudiante determinado puede recibir varios préstamos GSL, y es posible que cada uno tenga una tasa de interés y condiciones de subsidio diferentes.

Para apoyar el programa GSL, un grupo de distintas organizaciones cubre las funciones de prestamistas, fiadores y proveedores del servicio. Los estudiantes solicitan los préstamos a los prestamistas, entre los que se incluyen bancos, instituciones de ahorro y préstamos, y uniones de crédito. El Departamento de Educación de Estados Unidos hace posible los préstamos al garantizar el pago, siempre y cuando se cumplan ciertas condiciones. Los prestamistas se aseguran de que los solicitantes sean candidatos para el programa GSL. El proveedor de servicios registra las condiciones del estudiante, calcula el programa de pagos y realiza los cobros. El fiador se asegura de que los préstamos se manejen de manera apropiada supervisando el trabajo del proveedor de servicios. Si un préstamo entra en un estatus de moratoria (sin pago) y éste no cumple con los lineamientos del Departamento de Educación, el fiador puede ser el responsable. Para reducir el riesgo, por lo regular, los prestamistas no garantizan sus préstamos. En vez de ello, contratan a un proveedor de servicios y a un fiador.

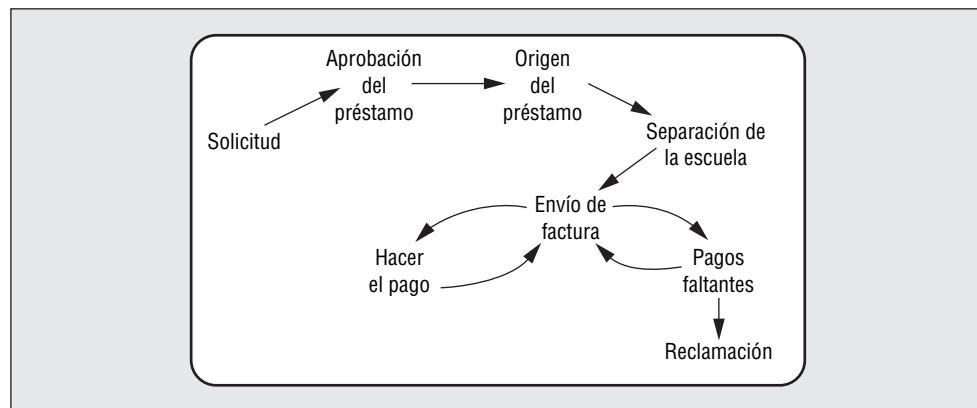
Student Loan Limited es un proveedor de servicios líder para GSL y otros tipos de préstamos para estudiantes. En la actualidad, Student Loan Limited utiliza un sistema heredado con una tecnología de archivos más antigua. La empresa quiere cambiar a una arquitectura cliente-servidor que utilice un DBMS relacional. La nueva arquitectura deberá permitirles responder con mayor facilidad a las nuevas regulaciones, así como buscar nuevos negocios, como el programa de préstamos directos.

### 13.1.2 Flujo de trabajo

El procesamiento de préstamos a los estudiantes sigue el patrón que muestra la figura 13.1. Los estudiantes solicitan un préstamo a un prestamista. En el proceso de aprobación, por lo regular, el prestamista identifica a un fiador. Si el préstamo se aprueba, el estudiante firma un pagaré que describe la tasa de interés y los términos de pago. Después de firmar el pagaré, el prestamista envía un formulario de origen del préstamo a Student Loan Limited. Después, Student Loan Limited desembolsa los fondos de acuerdo con los que se especifica en el formulario de origen del préstamo. Por lo general, los fondos se desembolsan en cada periodo de un año académico.

El proceso de pago comienza al salir de la escuela (por graduación o por abandono). Poco después de que el estudiante se separa del colegio, Student Loan Limited envía una carta de declaración que proporciona un cálculo del pago mensual requerido para cubrir el préstamo al final

**FIGURA 13.1**  
Flujo de trabajo para el procesamiento de préstamos



del periodo de pago. El estudiante recibe una carta de declaración por nota, excepto en el caso de que las notas se hayan consolidado en una sola. Las notas se consolidan si la tasa de interés, las condiciones de subsidio y el periodo de pago son similares.

Varios meses después de la separación, Student Loan Limited envía la primera factura. Para mayor comodidad, Student Loan Limited envía un estado de cuenta consolidado, aun cuando el estudiante tenga varios préstamos. Con la mayoría de los estudiantes, la compañía procesa facturas y pagos periódicos hasta que todos los préstamos quedan cubiertos. Si un estudiante se convierte en moroso, empiezan las actividades de cobro. Si el cobro tiene éxito, el estudiante regresa al ciclo de facturación-pago. De lo contrario, el préstamo entra a reclamación (falta de pago) y quizás pase a un despacho de cobros.

#### *Formulario de origen del préstamo*

El formulario de origen del préstamo, un documento electrónico que envía el prestamista, dispara la participación de Student Loan Limited. Las figuras 13.2 y 13.3 ilustran muestras de formularios con los datos del estudiante, el préstamo y el desembolso. Un formulario de origen incluye sólo un préstamo identificado para un número de préstamo único. Cada vez que se aprueba un préstamo, el prestamista envía un nuevo formulario de origen del préstamo. El método de desembolso puede ser mediante transferencia electrónica de fondos (EFT; *electronic funds transfer*) o por cheque. Si el método es por EFT (figura 13.2), es preciso proporcionar el número de ruta, el número de cuenta y la institución financiera. El plan de desembolso muestra la fecha de desembolso, la cantidad y cualquier cuota. Recuerde que el valor de la nota es la suma de las cantidades desembolsadas más las cuotas. Por lo regular, las cuotas son 6 por ciento del préstamo.

#### *Carta de declaración*

Después de que un estudiante se gradúa, pero antes de que empiece el pago del préstamo, Student Loan Limited debe enviar cartas de declaración para cada préstamo por pagar. Por lo regular, las cartas de declaración se envían aproximadamente 60 días después de que un estudiante se separa de la escuela. En algunos casos, se envía más de una carta de declaración por préstamos realizados en distintas ocasiones. Una carta de este tipo incluye campos para la cantidad del préstamo, la cantidad del pago mensual, el número de pagos, la tasa de interés, el cargo financiero total y la fecha del primer y último pago. En la carta de declaración de ejemplo (figura 13.4), los campos en el formulario están subrayados. Student Loan Limited tiene que conservar las copias de las cartas de declaración por si el fiador necesita revisar el procesamiento del préstamo a un estudiante.

#### *Estado de Cuenta*

Aproximadamente seis meses después de que un estudiante salió de la escuela, Student Loan Limited envía la primera factura. Para la mayoría de los estudiantes se enviarán facturas adicionales cada mes. En la terminología de Student Loan Limited, una factura se conoce como

**FIGURA 13.2**  
Muestra de fomulario  
de origen del pr  stamo

Loan Origination Form			
Loan No. L101	Date	6 Sept. 2004	
Student No.	\$100		
Name	Sam Student		
Address	15400 Any Street		
City, State, Zip	Anytown, USA 00999		
Phone (341) 555-2222	<b>Date of Birth</b> 11/11/1985		
Expected Graduation	May 2006		
Institution ID: U100	<b>Institution Name:</b> University of Colorado		
Address	1250 14th Street, Suite 700		
City, State, Zip	Denver CO 80217		
Disbursement Method	<i>EFT</i> <input checked="" type="checkbox"/> <i>Check</i>		
Routing No. R10001	<b>Account No.</b> A111000		
Disbursement Bank	Any Student Bank USA		
Lender No. LE100	<b>Lender Name</b> Any Bank USA		
Guarantor No. G100	<b>Guarantor Name</b> Any Guarantor USA		
Note Value: \$10000	Subsidized: Yes	<b>Rate:</b> 8.5%	
Disbursement Plan			
Date	Amount	Origination Fee	Guarantee Fee
30 Sept. 2004	\$3 200	\$100	\$100
30 Dec. 2004	\$3 200	\$100	\$100
30 Mar. 2005	\$3000	\$100	\$100

**FIGURA 13.3**  
Muestra de formu-  
lario de origen del  
pr  stamo

Loan Origination Form			
Loan No. L100	Date	7 Sept. 2005	
Student No.	\$100		
Name	Sam Student		
Address	15400 Any Street		
City, State, Zip	Anytown, USA 00999		
Phone (341) 555-2222	<b>Date of Birth</b> 11/11/1985		
Expected Graduation	May 2006		
Institution Id: U100	<b>Institution Name:</b> University of Colorado		
Address	1250 14th Street, Suite 700		
City, State, Zip	Denver CO 80217		
Disbursement Method	<i>EFT</i> <input type="checkbox"/> <i>Check</i> <input checked="" type="checkbox"/>		
Routing No. —	<b>Account No.</b> —		
Disbursement Bank	<b>Lender Name</b> Any Bank USA		
Lender No. LE100	<b>Guarantor Name</b> Any Guarantor USA		
Guarantor No. G100			
Note Value: \$10000	Subsidized: No	<b>Rate:</b> 8.0%	
Disbursement Plan			
Date	Amount	Origination Fee	Guarantee Fee
29 Sept. 2005	\$3 200	\$100	\$100
30 Dec. 2005	\$3 200	\$100	\$100
28 Mar. 2006	\$3000	\$100	\$100

**FIGURA 13.4**  
Muestra de carta de declaración

Disclosure Letter
1 July 2006
Subject: Loan L101
Dear Ms. Student,
According to our records, your guaranteed student loan enters repayments status in <u>September 2006</u> . The total amount that you borrowed was <u>\$10 000</u> . Your payment schedule includes <u>120</u> payments with an interest rate of <u>8.5%</u> . Your estimated finance charge is <u>\$4 877.96</u> . Your first payment will be due on <u>October 31, 2006</u> . Your monthly payment will be <u>\$246.37</u> . Your last payment is due <u>September 30, 2016</u> .
Sincerely,
Anne Administrator, Student Loan Limited

**FIGURA 13.5**  
Muestra de estado de cuenta para pagos con cheque

Statement of Account			
<b>Statement No.</b>	B100	<b>Date</b>	1 Oct. 2006
<b>Student No.</b>	S100	<b>Name</b>	Sam Student
<b>Street</b>	123 Any Street	<b>Zip</b>	00011
<b>City</b>	Any City	<b>State</b>	Any State
<b>Amount Due</b>	\$246.37	<b>Due Date</b>	31 Oct. 2006
<b>Payment Method</b>	<input checked="" type="checkbox"/> EFT	<b>Amount Enclosed</b>	
Loan Summary			
<b>Loan No.</b>	<b>Balance</b>	<b>Rate</b>	
L100	\$10 000	8.5%	
L101	\$10 000	8.2%	
<b>For Office Use Only</b>			
<b>Date Paid:</b>			

estado de cuenta. Las figuras 13.5 y 13.6 ilustran estados de cuenta muestra. La mitad superior del estado de cuenta contiene un número único de estado de cuenta, la cantidad que se debe, la fecha de pago, la cantidad pagada y el método de pago (EFT o cheque). Si el método de pago es por cheque (figura 13.5), el estudiante regresa el estado de cuenta a Student Loan Limited con el cheque adjunto. En este caso, la cantidad pagada la escribe el estudiante al regresar el estado de cuenta o el personal de captura de datos de Student Loan Limited al procesar el estado de cuenta. Si el método de pago es EFT (figura 13.6), la cantidad pagada aparece en el estado de cuenta con la fecha en que se va a realizar la transferencia. La fecha de pago la registra Student Loan Limited al recibir el pago. La mitad inferior de un estado de cuenta presenta las condiciones de cada préstamo. Para cada préstamo muestra el número de préstamo, el saldo pendiente y la tasa de interés.

Después de recibir el pago, Student Loan Limited aplica la cantidad principal del pago a los saldos pendientes. El pago se distribuye entre cada préstamo pendiente, de acuerdo con un programa de pagos asociado. Si un estudiante paga más de la cantidad especificada, la cantidad adicional se puede aplicar de varias formas; por ejemplo, se reduce primero el préstamo con la tasa de interés más alta, o bien, todos los préstamos pendientes se reducen en la misma cantidad. Las políticas del Departamento de Educación determinan el método para aplicar las cantidades

**FIGURA 13.6**  
Muestra de estado de cuenta para pago por EFT

Statement of Account					
Statement No.	B101	Date	1 Nov. 2006		
Student No.	S100	Name	Sam Student		
Street	123 Any Street	Zip	00011		
City	Any City	State	Any State		
Amount Due	\$246.37	Due Date	30 Nov. 2006		
Payment Method	Check <input checked="" type="checkbox"/> EFT <input type="checkbox"/>	Amount Enclosed			
Note: \$246.37 will be deducted from your account on 30 Nov. 2006					
Loan Summary					
Loan No.	Balance	Rate			
L100	\$9946.84	8.5%			
L101	\$9944.34	8.2%			
For Office Use Only					
Date Paid:					

**FIGURA 13.7**  
Muestra de reporte de actividad de un préstamo

Loan Activity Report			
Student No.	S100	Date	1 Feb. 2007
Street	123 Any Street	Name	Sam Student
City	Any City	Zip	00011
		State	Any State
Payment Summary for 2005			
Loan No.	Beg. Balance	Principal	Interest
L100	\$10000	160.60	211.37
L101	\$10000	168.12	203.85
For Office Use Only			
Date Paid:			

adicionales. Como sucede con la mayor parte de las políticas gubernamentales, ésta se encuentra sujeta a cambio. La aplicación de un pago a los saldos de los préstamos se pueden observar al comparar dos estados de cuenta consecutivos. Las figuras 13.5 y 13.6 muestran que 53.16 dólares del pago realizado en octubre de 2006 se aplicaron al préstamo L100.

#### *Reporte de actividad del préstamo*

Al final de cada año, Student Loan Limited envía a cada estudiante un reporte que resume toda la actividad del préstamo. Para cada uno, el reporte (figura 13.7) muestra el saldo del préstamo al iniciar el año, la cantidad total aplicada para reducir la cantidad principal, el total de intereses pagados y el saldo del préstamo al final del año. Student Loan Limited debe conservar copias de los reportes de actividad de los préstamos por si el fiador necesita revisar el procesamiento del préstamo a un estudiante.

### Nueva tecnología

Para reducir el papeleo, Student Loan Limited está interesado en crear imágenes de los documentos que necesitan los fiadores (cartas de declaración y reportes de actividad de los préstamos). Después de crear imágenes de los documentos, desean guardar los más recientes en la base de datos de préstamos a los estudiantes y los más antiguos en un archivo.

## 13.2 Modelado conceptual de datos

Las etapas del modelado conceptual de datos utiliza el planteamiento de integración incremental porque el caso no es muy extenso y los formularios son relacionados. La integración incremental empieza con el formulario de origen del préstamo porque dispara la participación de Student Loan Limited con un préstamo.

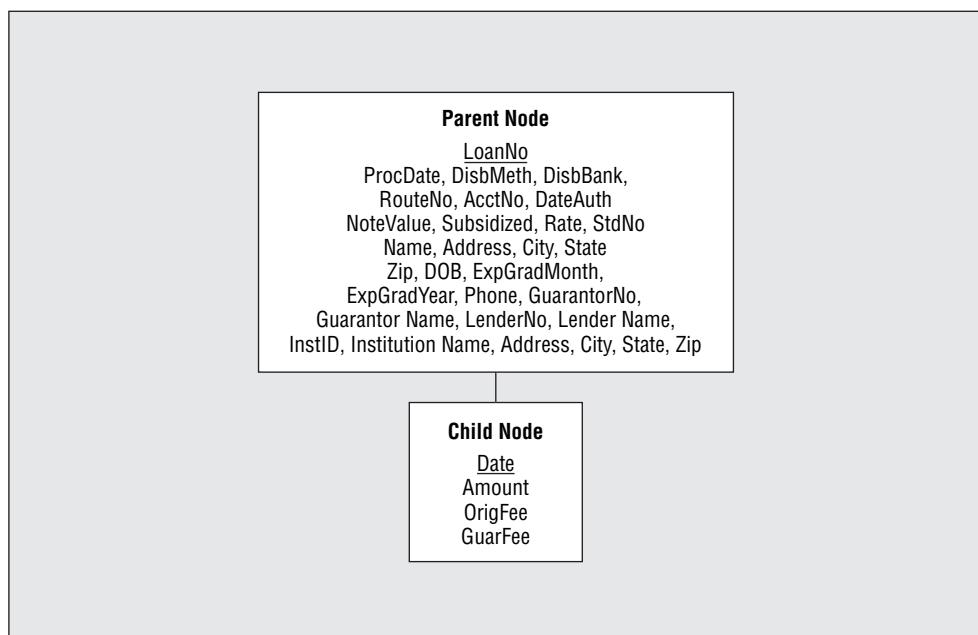
### 13.2.1 ERD para el formulario de origen del préstamo

El formulario de origen del préstamo contiene dos nodos, como muestra la figura 13.8. El nodo hija contiene los campos de desembolso que se repiten. El tipo de entidad *Loan* es el centro del ERD, como ilustra la figura 13.9. Los tipos de entidad circundantes (*Guarantor*, *Lender*, *Institution* y *Student*) y las relaciones asociadas se derivan del nodo madre. La cardinalidad mínima es 0 de *Loan* a *Guarantor* porque algunos préstamos no tienen un fiador (el prestamista representa ese papel). El tipo de entidad *DisburseLine* y la relación asociada se derivan del nodo hija. La tabla 13.1 muestra las suposiciones correspondientes a las anotaciones en la figura 13.9.

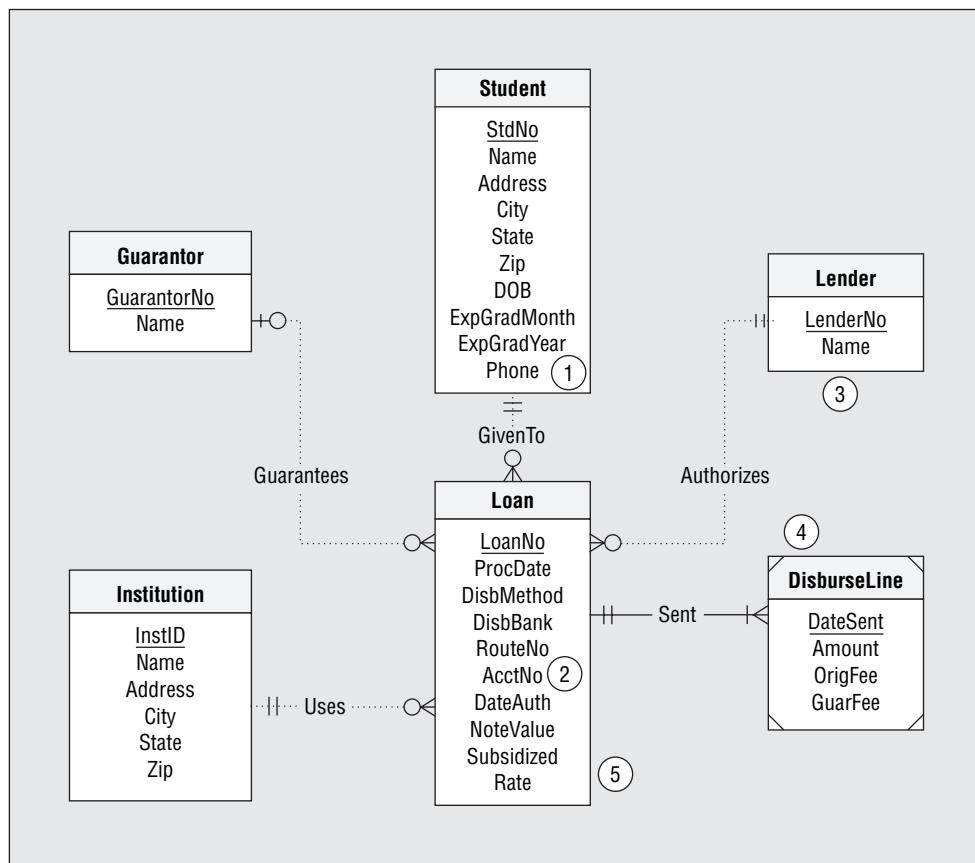
### 13.2.2 Integración incremental después de agregar una carta de declaración

La carta de declaración sólo contiene un nodo (figura 13.10) porque no tiene grupos que se repitan. La figura 13.11 muestra el ERD integrado, con las suposiciones correspondientes que muestra la tabla 13.2. El ERD en la figura 13.11 supone que las imágenes se pueden guardar en

**FIGURA 13.8**  
Estructura  
del formulario de  
origen de préstamo



**FIGURA 13.9**  
ERD para el formulario de origen del préstamo

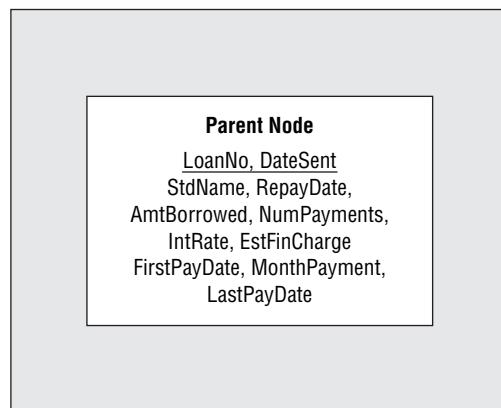


**TABLA 13.1**  
Suposiciones para el ERD en la figura 13.9

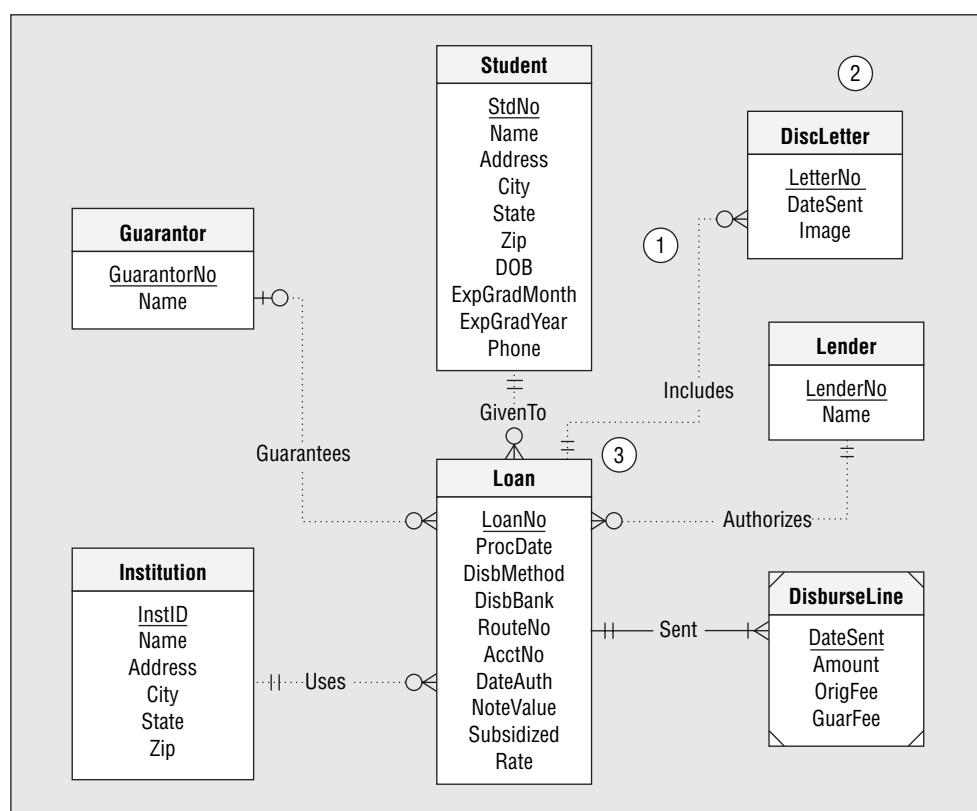
Número de anotación	Explicación
1	Los campos de la supuesta graduación esperada se pueden combinar en uno solo o conservarse como dos campos.
2	Si el método de desembolso es EFT son necesarios el número de ruta ( <i>RouteNo</i> ), el número de cuenta ( <i>AcctNo</i> ) y el banco que hará el desembolso ( <i>DisbBank</i> ) si el método de desembolso es EFT, de lo contrario, no se usan.
3	Quizás haya otros datos que guardar acerca de los prestamistas y fiadores. Como el formulario sólo muestra el número de identificación y el nombre, el ERD no incluye atributos adicionales.
4	<i>DisburseLine</i> depende de <i>Loan</i> para su identificación. Como <i>DisburseLine.DateSent</i> es una llave local, no puede haber dos desembolsos del mismo préstamo en la misma fecha. La clave primaria de <i>DisburseLine</i> es una concatenación de <i>LoanNo</i> y <i>DateSent</i> .
5	La suma de la cantidad más la cuota de origen y la cuota de fianza en el plan de desembolso debe ser igual al valor de la nota.

la base de datos. Por lo tanto, no se guardan los campos particulares de una carta de declaración. El campo único *LetterNo* se agregó como identificador de una carta de declaración. Si las imágenes no pueden almacenarse en la base de datos, tal vez sea necesario guardar algunos de los campos en la carta de declaración porque son difíciles de calcular.

**FIGURA 13.10**  
Estructura de la carta de declaración



**FIGURA 13.11**  
ERD después de agregar la carta de declaración



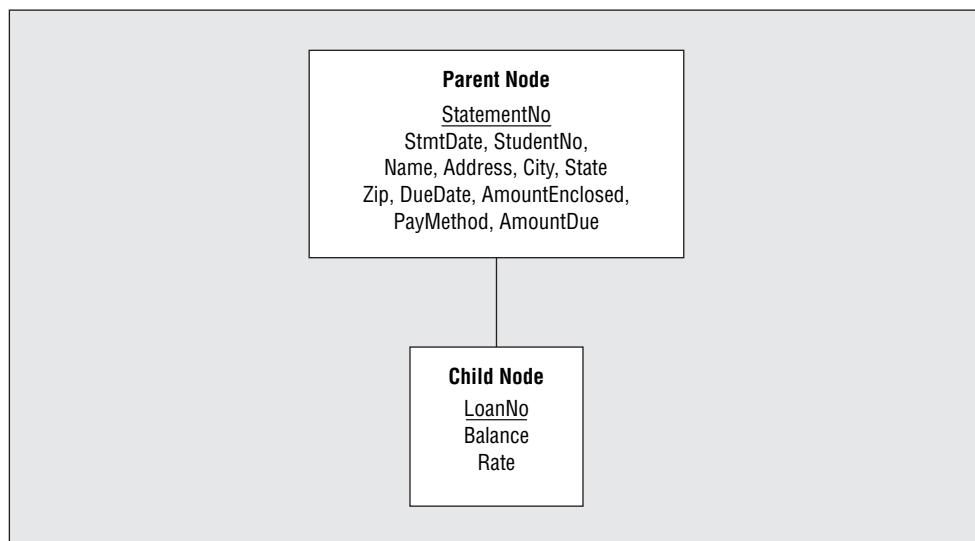
**TABLA 13.2**  
Suposiciones para el ERD en la figura 13.11

Número de anotación	Explicación
1	La relación entre <i>DiscLetter</i> y <i>Loan</i> permite varias cartas por préstamo. Como se indica en el caso, es posible mandar varias cartas de declaración para el mismo préstamo.
2	El campo <i>Image</i> contiene una imagen escaneada de la carta. Es probable que el fiador necesite una copia de la carta si el préstamo se somete a una auditoría. En caso de no emplear una tecnología de imágenes y como una alternativa al guardado de la imagen, es posible guardar un indicador de la ubicación física, en caso de no emplear una tecnología de imágenes.
3	Es necesaria una cardinalidad mínima de 0 porque no se crea un plan de pagos sino hasta que el estudiante sale de la escuela.

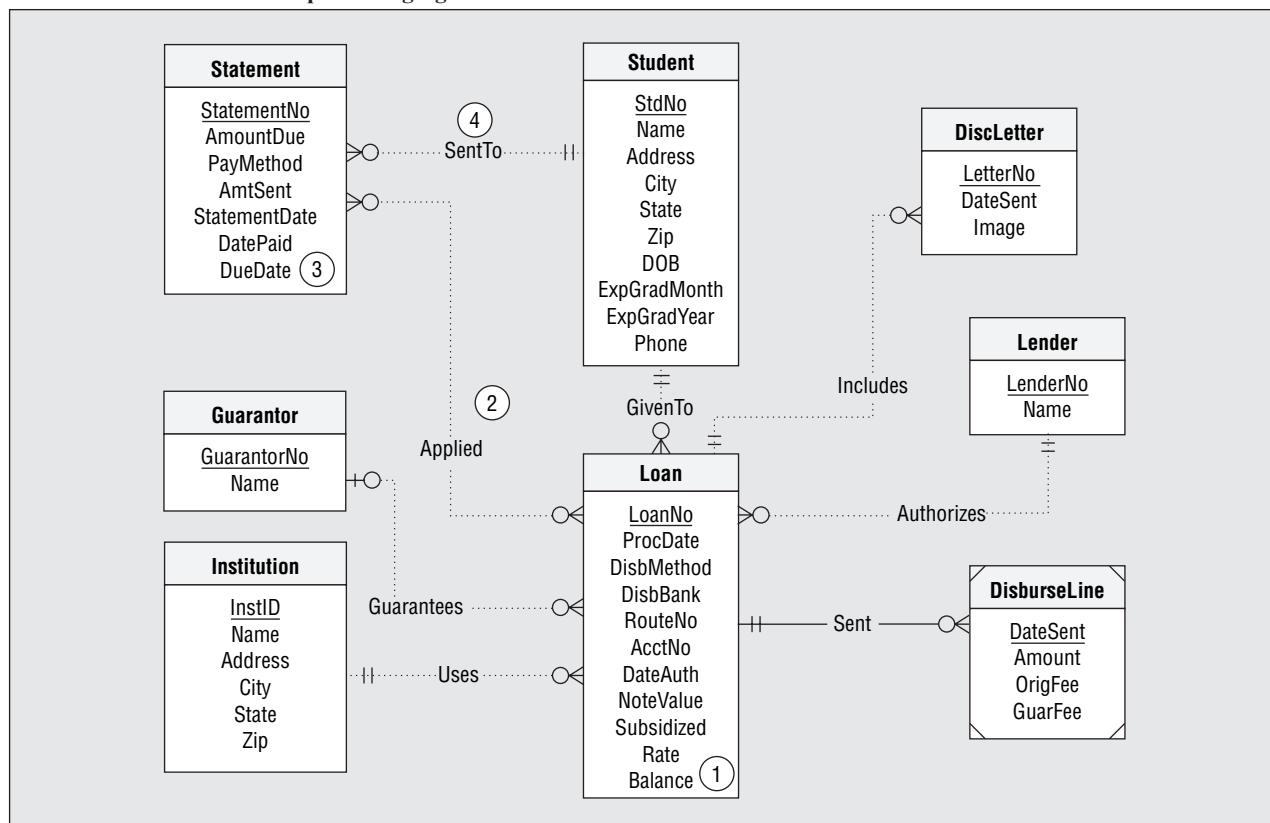
### 13.2.3 Integración incremental después de agregar el estado de cuenta

El estado de cuenta contiene nodos madre e hija (figura 13.12) porque tiene un grupo que se repite. La figura 13.13 muestra el ERD integrado con las suposiciones correspondientes que se muestran en la tabla 13.3. La relación *Applied* en la figura 13.13 representa la relación madre-hija en

**FIGURA 13.12**  
Estructura del estado de cuenta



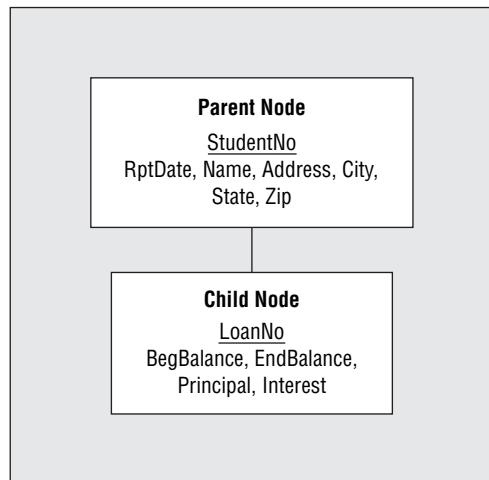
**FIGURA 13.13** ERD después de agregar el estado de cuenta



**TABLA 13.3**  
Suposiciones para el ERD en la figura 13.13

Número de anotación	Explicación
1	<i>Balance</i> se agrega como campo para reflejar el resumen del préstamo en un estado de cuenta. El saldo refleja el último pago hecho a un préstamo.
2	La relación <i>Applied</i> se crea al mismo tiempo que el saldo. Sin embargo, los campos de la cantidad principal y el interés no se actualizan sino hasta que se recibe el pago. Los atributos ( <i>Principal</i> , <i>Interest</i> , <i>CumPrincipal</i> y <i>CumInterest</i> ) de la relación aplicada no se muestran en el diagrama con el fin de reducir el hacinamiento. <i>CumPrincipal</i> y <i>CumInterest</i> son columnas derivadas que facilitan el reporte de actividad del préstamo.
3	Si el método de pago es EFT, tal vez sean necesarios otros atributos en <i>Statement</i> , como número de ruta y número de cuenta. Como estos atributos no se muestran en un estado de cuenta, se omiten del tipo de entidad <i>Statement</i> .
4	La relación <i>SentTo</i> es redundante. Se puede derivar de las relaciones <i>Applied</i> y <i>GivenTo</i> . Si el tiempo para derivar la relación <i>SentTo</i> no es oneroso, es posible dejarla.

**FIGURA 13.14**  
Estructura del reporte de actividad del préstamo

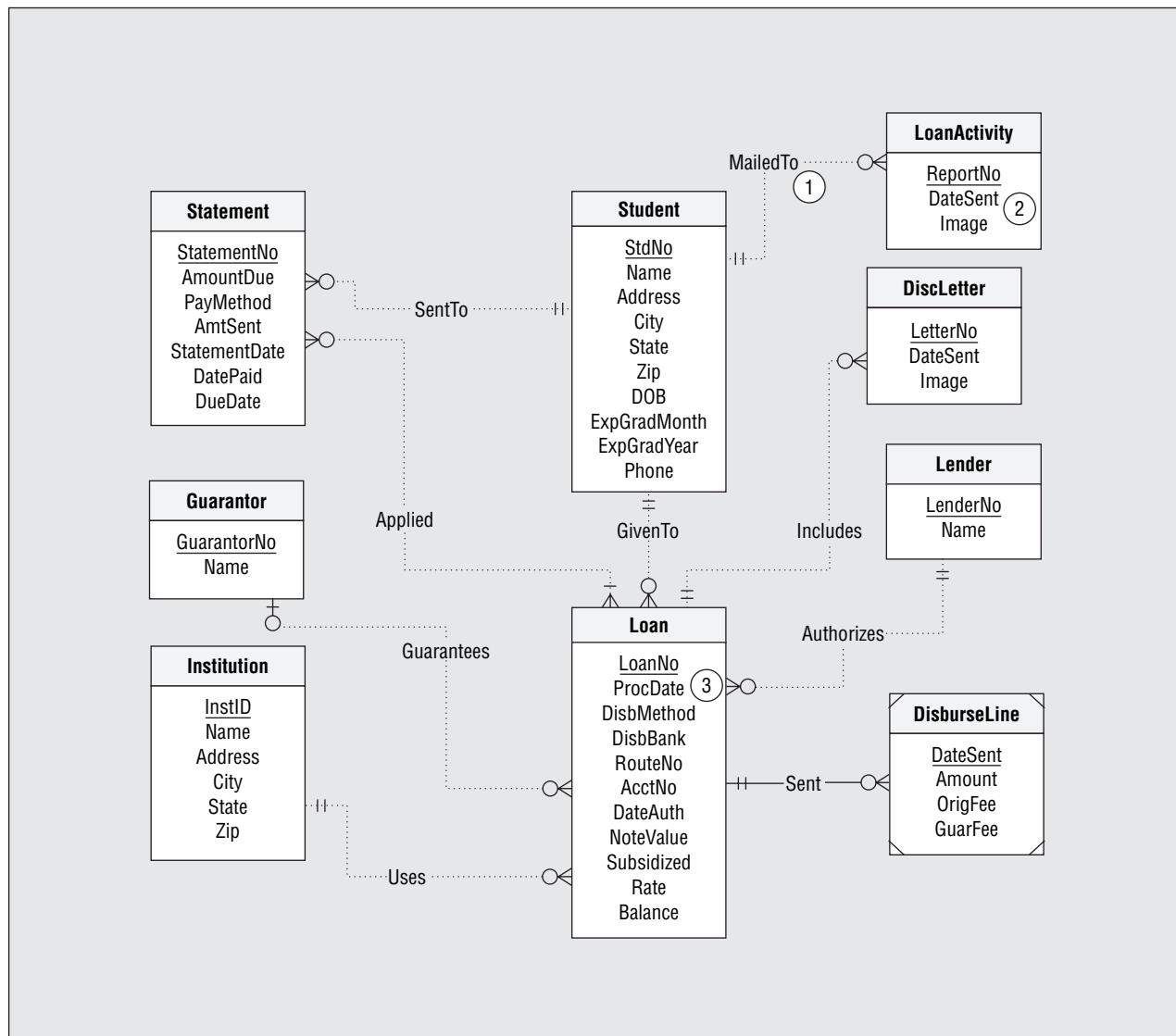


la jerarquía del formulario. La cardinalidad mínima es 0 de *Loan* a *Statement* porque un préstamo no tiene ninguna cantidad aplicada hasta después de que entra en el estatus de pago.

### 13.2.4 Integración incremental después de agregar el reporte de actividad del préstamo

El reporte de actividad del préstamo contiene nodos madre e hija (figura 13.14), porque tiene un grupo que se repite. La figura 13.15 muestra el ERD integrado con las suposiciones correspondientes en la tabla 13.4. Al igual que el ERD para la carta de declaración (figura 13.11), el ERD en la figura 13.15 supone que las imágenes se pueden guardar en la base de datos. Por tanto, no se guardan los campos particulares en el reporte de actividades del préstamo. El campo único *ReportNo* se agregó como identificador de un reporte de actividad. Si las imágenes no se pueden guardar en la base de datos, quizás sea necesario guardar algunos de los campos en el reporte de actividad del préstamo porque son difíciles de calcular.

FIGURA 13.15 ERD después de agregar el reporte de actividad del préstamo

TABLA 13.4  
Suposiciones para  
el ERD en la figura  
13.15

Número de anotación	Explicación
1	El tipo de entidad <i>LoanActivity</i> no está directamente relacionado con el tipo de entidad <i>Loan</i> , porque se supone que un reporte de actividad resume todos los préstamos de un estudiante.
2	El campo <i>Image</i> contiene una imagen escaneada del reporte. Es probable que el fiador necesite una copia del informe en caso de que el préstamo se someta a auditoría. Como una alternativa al almacenamiento de la imagen, es posible guardar un indicador de la ubicación física, si no se emplea una tecnología de imágenes.
3	Para facilitar los cálculos se pueden agregar, los campos para la cantidad principal anual y el interés se podrían agregar al tipo de entidad <i>Loan</i> . Estos campos se pueden actualizar después de recibir cada pago, y se deben considerar durante el diseño físico de la base de datos.

## 13.3 Refinamiento del esquema conceptual

Después de crear un ERD conceptual, debe refinarlo aplicando las reglas de conversión para producir un diseño de tabla inicial, así como también debe utilizar reglas de normalización para eliminar el exceso de redundancias del diseño de la tabla inicial. Esta sección describe las mejoras al ERD conceptual que producen un buen diseño de tabla para Student Loan Limited.

### 13.3.1 Conversión del esquema

La conversión se puede llevar a cabo utilizando las primeras cuatro reglas (capítulo 6) como se presentan en la tabla 13.5. A la interrelación *Guarantees* se le podría aplicar la regla opcional de relación 1-M (regla 5); sin embargo, el número de préstamos sin fiadores parece reducido, de modo que se utiliza la regla de relación 1-M. No es necesaria la regla de jerarquía de generalización (regla 6) porque el ERD (figura 13.9) no tiene ninguna jerarquía de generalización. El resultado de la conversión se muestra en la tabla 13.6 (las llaves primarias aparecen subrayadas y las foráneas con letras cursivas) y en la figura 13.16 (representación gráfica de tablas, llaves primarias y llaves foráneas).

**TABLA 13.5**  
Reglas utilizadas para convertir el ERD de la figura 13.15

Regla de conversión	Objetos	Comentarios
Regla de tipo de entidad	Tablas <i>Student</i> , <i>Statement</i> , <i>Loan</i> , <i>DisclLetter</i> , <i>LoanActivity</i> , <i>Lender</i> , <i>Guarantor</i> , <i>Institution</i> , <i>DisburseLine</i>	Las llaves primarias en cada tabla son idénticas a los tipos de entidad, excepto <i>DisburseLine</i> .
Regla de relación 1-M	<i>Loan.StdNo</i> , <i>Loan.GuarantorNo</i> , <i>Loan.LenderNo</i> , <i>LoanActivity</i> , <i>StdNo</i> , <i>DisclLetter.LoanNo</i> , <i>Statement.StdNo</i> , <i>DisburseLine</i> , <i>LoanNo</i> , <i>Loan.InstID</i>	Se agregan las columnas con las llaves foráneas y las limitaciones a la integridad referencial.
Regla de relación M-N	Tabla <i>Applies</i>	Llave primaria combinada: <i>StatementNo</i> , <i>LoanNo</i>
Regla de dependencia de identificación	Llave primaria ( <i>LoanNo</i> , <i>DateSent</i> )	<i>LoanNo</i> agregado a la llave primaria de la tabla <i>DisburseLine</i> .

**TABLA 13.6**  
Lista de tablas después de la conversión

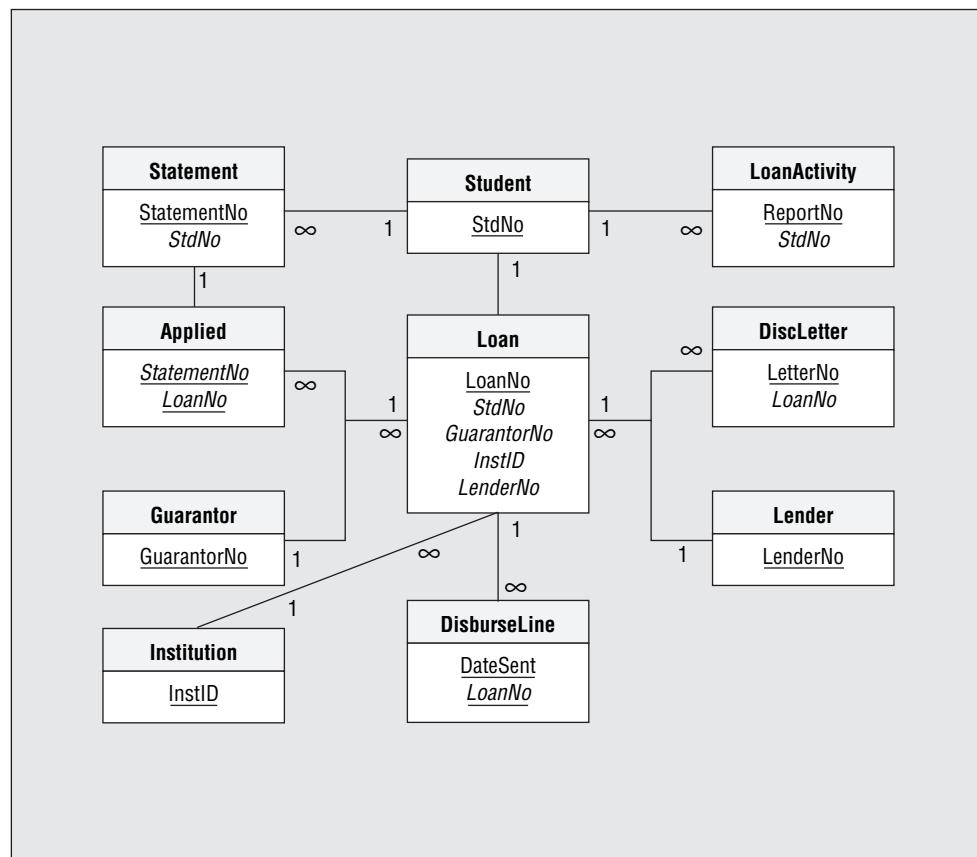
```

Student(StdNo, Name, Address, Phone, City, State, Zip, ExpGradMonth, ExpGradYear, DOB)
Lender(LenderNo, Name)
Guarantor(GuarantorNo, Name)
Institution(InstID, Name, Address, City, State, Zip)
Loan(LoanNo, ProcDate, DisbMethod, DisbBank, RouteNo, AcctNo, DateAuth, NoteValue, Subsidized, Rate, Balance, StdNo, InstID, GuarantorNo, LendNo)
DisclLetter(LetterNo, DateSent, Image, LoanNo)
LoanActivity(ReportNo, DateSent, Image, StdNo)
DisburseLine(LoanNo, DateSent, Amount, OrigFee, GuarFee)
Statement(StatementNo, AmtDue, PayMethod, AmtSent, StatementDate, DatePaid, DueDate, StdNo)
Applied(LoanNo, StatementNo, Principal, Interest, CumPrincipal, CumInterest)

```

**FIGURA 13.16**

Diagrama del modelo relacional para el diseño de la tabla inicial



**TABLA 13.7**  
Lista de FD

Tabla	FD
Student	StdNo → Name, Address, City, State, Zip, ExpGradMonth, ExpGradYear, DOB, Phone; Zip → State
Lender	LenderNo → Name
Guarantor	GuarantorNo → Name
Institution	InstlID → Name, Address, City, State, Zip; Zip → City, State
Loan	LoanNo → ProcDate, DisbMethod, DisbBank, RouteNo, AcctNo, DateAuth, NoteValue, Subsidized, Rate, Balance, StdNo, InstlID, GuarantorNo, LenderNo; RouteNo → DisbBank
DiscLetter	LetterNo → DateSent, Image, LoanNo; LoanNo, DateSent → LetterNo, Image
LoanReport	ReportNo → DateSent, Image, StdNo; StdNo, DateSent → ReportNo, Image
DisburseLine	LoanNo, DateSent → Amount, OrigFee, GuarFee
Statement	StatementNo → AmtDue, PayMethod, AmtSent, StatementDate, DatePaid, DueDate, StdNo
Applied	LoanNo, StatementNo → Principal, Interest, CumPrincipal, CumInterest

### 13.3.2 Normalización

Las tablas que resultan del proceso de conversión podrían tener redundancia. Para eliminarla, es preciso incluir los FD para cada tabla y aplicar las reglas de normalización o el procedimiento de síntesis simple. La tabla 13.7 presenta los FD para cada tabla.

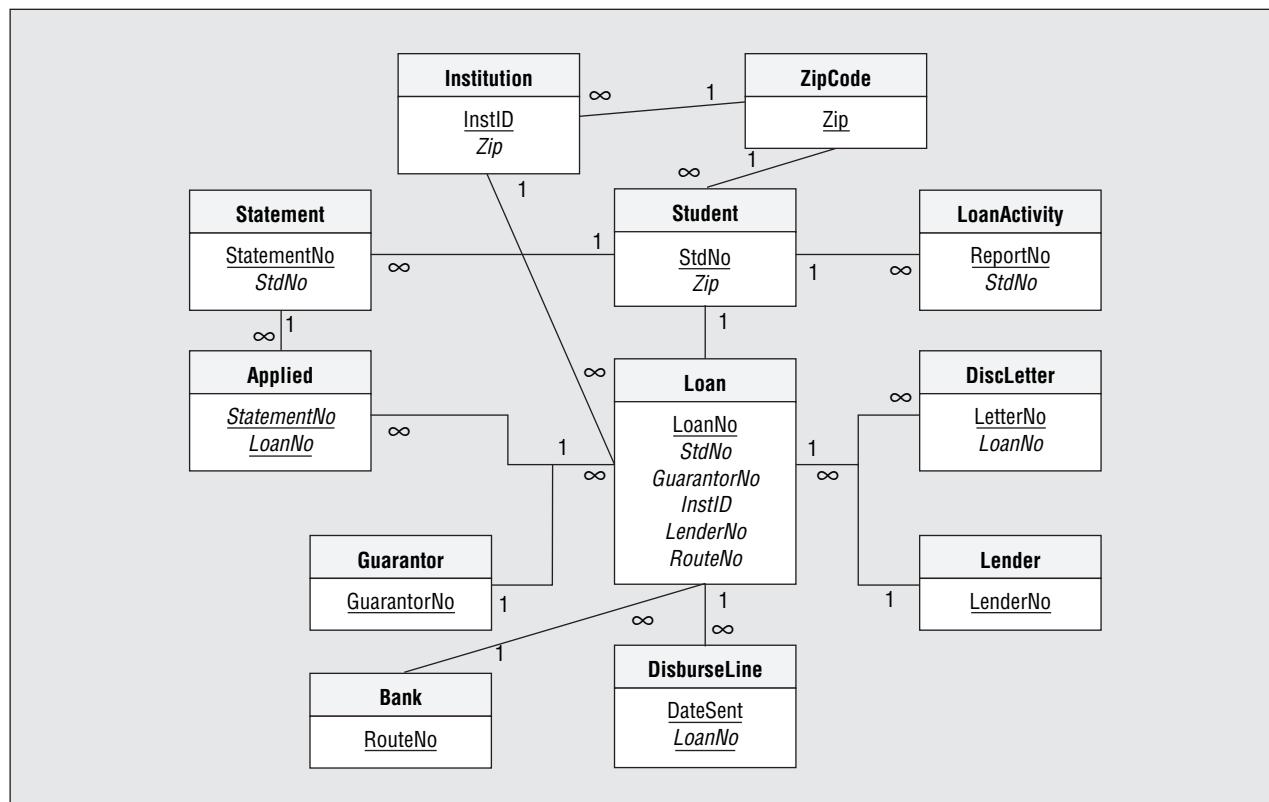
Como la mayoría de los FD comprenden una llave primaria del lado izquierdo, no hay mucho trabajo de normalización. Sin embargo, las tablas *Loan*, *Student* e *Institution* violan el BCNF, ya que tienen determinantes que no son llaves candidatas. Las tablas *DiscLetter* y *LoanReport* no

violan el BCNF porque todos los determinantes son llaves candidatas. Para las tablas que violan el BCNF, presentamos las explicaciones y opciones acerca de la división de las tablas:

- *Student* no está en BCNF debido al FD con *Zip*. Si Student Loan Limited quiere actualizar los códigos postales de manera independiente de los estudiantes, es necesario agregar una tabla por separado.
- *Loan* no está en BCNF debido al FD que comprende *RouteNo*. Si Student Loan Limited quiere actualizar los bancos de manera independiente de los préstamos, es preciso crear una tabla por separado.
- *Institution* no está en BCNF debido a los FD con *Zip*. Si Student Loan Limited desea actualizar los códigos postales de manera independiente de las instituciones, es necesario agregar una tabla por separado. Sólo se necesita una tabla con códigos postales para *Student* e *Institution*.

En el diseño de la tabla revisada (figura 13.17), se agregan las tablas *ZipCode* y *Bank* para eliminar las redundancias. El apéndice 13.B muestra enunciados CREATE TABLE con la lista de tablas revisadas. Asimismo, incluye las acciones de eliminar y actualizar. Para la mayor parte de las llaves foráneas, las eliminaciones están limitadas porque las tablas madre e hija correspondientes no están estrechamente relacionadas. Por ejemplo, las eliminaciones están limitadas para la llave foránea *Loan.InstID* porque las tablas *Institution* y *Loan* no están estrechamente relacionadas. En contraste, las eliminaciones se realizan en cascada para la llave foránea *DisburseLine.LoanNo* porque las líneas de desembolso dependen de la identificación en el préstamo relacionado. Las eliminaciones también se hacen en cascada para la llave foránea *Applied.StatementNo* porque las filas aplicadas representan líneas de enunciados que no tienen ningún significado sin el enunciado. La acción de actualización de la mayor parte de las llaves foráneas se programó para realizarse en cascada con el fin de permitir el cambio fácil de los valores de las llaves primarias.

**FIGURA 13.17** Diagrama de modelo relacional para el diseño de la tabla revisada



## 13.4 Diseño físico de la base de datos y desarrollo de aplicaciones

Después de producir un buen diseño de tablas, está listo para implementar la base de datos. Esta sección describe las decisiones en el diseño físico de la base de datos, incluyendo selección del índice, datos derivados y desnormalización para la base de datos de Student Loan Limited. Antes de describir estas decisiones, definimos los perfiles de tablas y aplicaciones. Después de presentar las decisiones en el diseño físico de bases de datos, ilustramos el desarrollo de aplicaciones de algunos formularios y reportes como verificación cruzada en el desarrollo de bases de datos.

### 13.4.1 Perfiles de aplicaciones y tablas

Para aclarar el uso anticipado de la base de datos, los documentos que describimos en la sección 13.1 se dividen en aplicaciones de acceso a las bases de datos, como resume la tabla 13.8. Tres aplicaciones independientes se asocian con el formulario de origen del préstamo. La verificación de los datos comprende recuperaciones para asegurarse de que existan el estudiante, el prestamista, la institución y el fiador. Si un estudiante no existe, se agrega una nueva fila. La creación de un préstamo comprende la inserción de una fila en la tabla *Loan* y de varias filas en la tabla *DisburseLine*. Para los otros documentos, hay una aplicación para crear y recuperar el documento. También existe una aplicación para actualizar las tablas *Applied* y *Loan* en los estados de cuenta al recibir los pagos.

Para tomar decisiones sobre el diseño físico de bases de datos, es necesario especificar la importancia relativa de las aplicaciones. Las frecuencias en la tabla 13.9 suponen 100 000 préstamos nuevos por año y 100 000 estudiantes cubriendo sus préstamos cada año. Las aplicaciones del origen de préstamo y las aplicaciones del estado de cuenta dominan la carga de trabajo. Las frecuencias (por año) son suficientes para indicar la importancia relativa de las aplicaciones. Tal vez sea necesaria una mejor especificación (por ejemplo, por mes o día) para programar el trabajo, como arreglar las cosas para un procesamiento por lote en lugar de un procesamiento en línea. Por ejemplo, las aplicaciones que comprenden formularios de origen de préstamo se pueden procesar por lote, en vez de hacerlo en línea.

**TABLA 13.8 Características de las aplicaciones**

Aplicación	Tablas	Condiciones
Verificar los datos (para el origen de préstamo)	<i>Student, Lender, Institution, Guarantor</i>	<i>StdNo = \$X; LenderNo = \$Y; InstID = \$Z; GuarantorNo = \$W</i>
Crear préstamo (para origen de préstamo)	<i>Loan, DisburseLine</i>	Una fila insertada en <i>Loan</i> ; varias filas insertadas en <i>DisburseLine</i>
Crear estudiante (para origen del préstamo)	<i>Student</i>	Una fila insertada
Crear carta de declaración	<i>Student, Loan, DiscLetter</i>	Insertar fila en <i>DiscLetter</i> ; recuperar filas para <i>Student</i> y <i>Loan</i> ( <i>LoanNo = \$X</i> )
Mostrar carta de declaración	<i>DiscLetter</i>	<i>LoanNo = \$X</i>
Crear reporte de actividad del préstamo	<i>Student, Loan, LoanActivity, Applied, Statement</i>	Insertar fila <i>LoanActivity</i> ; recuperar filas de <i>Student</i> ( <i>StdNo = \$X</i> ) y <i>Loan Statement</i> ( <i>LoanNo = \$X DatePaid</i> en el año pasado)
Mostrar el reporte de actividad de préstamo	<i>LoanActivity</i>	<i>StdNo = \$X</i>
Crear estado de cuenta	<i>Statement</i>	Una fila insertada en <i>Statement</i> ; varias filas insertadas en <i>Applied</i>
Mostrar estado de cuenta	<i>Statement, Student, Applied, Loan</i>	<i>StdNo = \$X AND DateSent = \$Y</i> ; en algunas ocasiones con el uso de <i>StatementNo = \$Z</i>
Aplicar pago	<i>Applied, Statement, Loan</i>	Renglones de <i>Applied</i> actualizados; <i>LoanNo = \$X AND StatementNo = \$Y</i> ; <i>Balance</i> actualizado en la tabla <i>Loan</i>

**TABLA 13.9**  
Frecuencias de la aplicación

Aplicación	Frecuencia	Comentarios
Verificar datos	100 000/año	La mayor parte de la actividad al principio del periodo
Crear préstamo	100 000/año	La mayor parte de la actividad al principio del periodo
Crear estudiante	20 000/año	La mayoría de los estudiantes son repetidos
Crear carta de declaración	50 000/año	Distribuir de manera uniforme en todo lo largo del año
Mostrar carta de declaración	5 000/año	Distribuir de manera uniforme en todo lo largo del año
Crear reporte de actividad de préstamo	30 000/año	Procesamiento al final del año
Mostrar reporte de actividad de préstamo	5 000/año	Distribuir de manera uniforme en todo lo largo del año
Crear estado de cuenta	100 000/año	Una vez al mes
Mostrar estado de cuenta	10 000/año	Distribuir de manera uniforme en todo el año
Aplicar el pago	100 000/año	Distribuir de manera uniforme en todo el mes

**TABLA 13.10**  
Perfiles de tablas

Tabla	Número de filas	Columna (número de valores únicos)
<i>Student</i>	100 000	<i>StdNo</i> (PK), <i>Name</i> (99 000), <i>Address</i> (90 000), <i>City</i> (1 000), <i>Zip</i> (1 000), <i>DOB</i> (365), <i>ExpGradMonth</i> (12), <i>ExpGradYear</i> (10)
<i>Loan</i>	300 000	<i>LoanNo</i> (PK), <i>ProcDate</i> (350), <i>DisbMethod</i> (3), <i>DisbBank</i> (3 000), <i>RouteNo</i> (3 000), <i>AcctNo</i> (90 000), <i>DateAuth</i> (350), <i>NoteValue</i> (1 000), <i>Subsidized</i> (2), <i>Rate</i> (1 000), <i>Balance</i> (10 000), <i>StdNo</i> (100 000), <i>InstID</i> (2 000), <i>GuarantorNo</i> (100), <i>LenderNo</i> (2 000)
<i>Institution</i>	2 000	<i>InstID</i> (PK), <i>Name</i> (2 000), <i>Address</i> (2 000), <i>City</i> (500), <i>State</i> (50), <i>Zip</i> (500)
<i>DisLetter</i>	1 000 000	<i>LetterNo</i> (PK), <i>DateSent</i> (350), <i>Image</i> (1 000 000), <i>LoanNo</i> (300 000)
<i>Statement</i>	2 000 000	<i>StatementNo</i> (PK), <i>AmtDue</i> (100 000), <i>PayMethod</i> (3), <i>AmtSent</i> (200 000), <i>StatementDate</i> (350), <i>DatePaid</i> (350), <i>DueDate</i> (350), <i>StdNo</i> (100 000)
<i>Guarantor</i>	100	<i>GuarantorNo</i> (PK), <i>Name</i> (100)
<i>Bank</i>	3 000	<i>RouteNo</i> (PK), <i>DisbBank</i> (3 000)
<i>DisburseLine</i>	900 000	<i>LoanNo</i> (300 000), <i>DateSent</i> (350), <i>Amount</i> (5 000), <i>OrigFee</i> (5 000), <i>GuarFee</i> (5 000)
<i>Applied</i>	6 000 000	<i>LoanNo</i> (300 000), <i>StatementNo</i> (2 000 000), <i>Principal</i> (100 000), <i>Interest</i> (1 000 000)
<i>ZipCode</i>	1 000	<i>Zip</i> (PK), <i>State</i> (50)
<i>Lender</i>	2 000	<i>LenderNo</i> (PK), <i>Name</i> (2 000)

Después de definir los perfiles de aplicaciones, es posible definir los perfiles de tablas. El volumen de la actividad de modificación ( inserciones, actualizaciones, eliminaciones) puede ayudar en el cálculo de los perfiles de las tablas. Además, deberá utilizar estadísticas de los sistemas y entrevistas existentes con el personal de aplicaciones clave para realizar los cálculos. La tabla 13.10 ofrece un panorama general de los perfiles. Es posible agregar más detalles acerca de las distribuciones de columnas y relaciones después de que el sistema esté parcialmente poblado.

### 13.4.2 Selección de índices

Es posible seleccionar los índices utilizando los perfiles de aplicaciones y las reglas descritas en el capítulo 8. Para aclarar el proceso de selección, vamos a considerar las necesidades de

recuperación antes de las de manipulación. Recuerde que las reglas de la 1 a la 5 (capítulo 8) comprenden la selección de índices para las necesidades de recuperación. La lista siguiente estudia las opciones de índices útiles para propósitos de recuperación.

- Los índices en las llaves primarias de las tablas *Student*, *Loan*, *Guarantor*, *Institution*, *DiscLetter*, *LoanActivity*, *Statement* y *Bank* ofrecen soporte para verificar el préstamo, mostrar la carta de declaración, mostrar el reporte de actividad y mostrar aplicaciones del estado de cuenta.
- Un índice sin *cluster* sobre el nombre del estudiante puede ser una buena opción para soportar la recuperación de estados de cuenta y los reportes de actividad de préstamo.
- Para soportar los *joins*, pueden ser útiles los índices sin *cluster* en las llaves foráneas *Loan*.*StdNo*, *Statement.StdNo*, *Applied.LoanNo* y *Applied.StatementNo*. Por ejemplo, un índice en *Loan.StdNo* facilita el enlace de las tablas *Student* y *Loan* cuando se tiene un valor *StdNo* específico.

Como las tablas *Applied* y *Loans* tienen muchas modificaciones, es necesario proceder con precaución con los índices sobre los campos de componentes. Sin embargo, algunos factores podrían compensar el impacto de la actividad de modificación. Las actualizaciones en la aplicación para aplicar pagos no afectan los campos de llaves foráneas en estas tablas. El procesamiento en lote puede reducir el impacto de las inserciones en las tablas *Loan* y *Applied*. Las aplicaciones para crear préstamo y crear estado de cuenta se pueden realizar en lote porque los formularios de origen de préstamo se reciben en lote y los estados de cuenta se pueden producir en lote. Si los índices representan una carga muy pesada para el procesamiento en lote, es posible dejar los índices antes del procesamiento y volver a crearlos cuando éste termine.

La tabla 13.11 muestra opciones de índices con base en la discusión anterior. La opción supone que los índices de llaves foráneas en las tablas *Applied* y *Loan* no dificulta la actividad de inserción. Es probable que se requiera una investigación posterior para determinar el impacto de los índices en las inserciones de las tablas *Loan* y *Applied*.

### 13.4.3 Datos derivados y decisiones de desnormalización

Existen algunos datos derivados en el diseño de las tablas revisadas. Las columnas *CumPrincipal* y *CumInterest* se derivan de la tabla *Applied*. Las tablas *DiscLetter* y *LoanActivity* tienen muchos datos derivados en las columnas *Image*. En todos estos casos, los datos derivados parecen justificados debido a la dificultad de calcularlos.

**TABLA 13.11**  
Selecciones de índices para el diseño de tablas revisadas

Columna	Tipo de Índice	Regla
<i>Student.StdNo</i>	Clustering	1
<i>Student.Name</i>	Nonclustering	3
<i>Statement.StatementNo</i>	Clustering	1
<i>DiscLetter.LetterNo</i>	Clustering	1
<i>Loan.LoanNo</i>	Clustering	1
<i>Institution.InstID</i>	Clustering	1
<i>Guarantor.GuarantorNo</i>	Clustering	1
<i>Lender.LenderNo</i>	Clustering	1
<i>LoanActivity.ReportNo</i>	Clustering	1
<i>ZipCode.Zip</i>	Clustering	1
<i>Bank.RouteNo</i>	Clustering	1
<i>Statement.StdNo</i>	Nonclustering	2
<i>Loan.StdNo</i>	Nonclustering	2
<i>Applied.StatementNo</i>	Clustering	2
<i>Applied.LoanNo</i>	Nonclustering	2

La desnormalización puede ser útil para algunas llaves foráneas. Si los usuarios piden con frecuencia el nombre con la llave foránea, la desnormalización puede ser útil para las llaves foráneas de la tabla *Loan*. Por ejemplo, el hecho de guardar *LenderNo* y *Lender.Name* en la tabla *Loan* viola el BCNF, pero puede reducir los enlaces entre las tablas *Loan* y *Lender*. El uso de la base de datos se podría supervisar con detenimiento para determinar si la tabla *Loan* se debe desnormalizar agregando columnas de nombres, además de las columnas *LenderNo*, *GuarantorNo*, *InstID* y *RouteNo*. La desnormalización es una buena idea si puede mejorar el desempeño de manera significativa, ya que las tablas *Lender*, *Guarantor*, *Institution* y *Bank* son relativamente estáticas.

#### 13.4.4 Otras decisiones de implementación

Hay varias decisiones de implementación que comprenden el proceso de desarrollo de bases de datos. En esta sección destacamos este tipo de decisiones porque tienen un impacto significativo en el éxito del sistema de servicio de préstamos.

- La conversión del sistema antiguo al nuevo sin problemas es un importante aspecto. Un impedimento para esta conversión sin problemas son los volúmenes de procesamiento. En ocasiones, los volúmenes de procesamiento en un sistema nuevo pueden ser mucho más elevados que en el antiguo. Una forma de reducir los potenciales problemas de desempeño es ejecutar el sistema antiguo y nuevo en paralelo, cambiando a más trabajo al nuevo sistema después de un tiempo.
- Una parte importante del proceso de conversión comprende los datos antiguos. Por lo general, no es difícil convertir los datos antiguos al nuevo formato, a excepción de las preocupaciones por la calidad de los datos. En ocasiones, la mala calidad de los datos antiguos provoca muchos rechazos en el proceso de conversión. Este proceso debe ser sensible al rechazo de datos de mala calidad porque es probable que los rechazos requieran de numerosas correcciones manuales.
- El tamaño de los datos en forma de imagen (reportes de actividad de préstamos y cartas de declaración) puede tener un impacto en el desempeño de la base de datos. El hecho de archivar los datos en forma de imagen puede mejorar el desempeño para aquellas imágenes que se recuperen con poca frecuencia.

#### 13.4.5 Desarrollo de aplicaciones

Para completar el desarrollo de la base de datos, es necesario implementar los formularios y reportes utilizados en los requerimientos de diseño de la base. El hecho de implementar los formularios y reportes ofrece una revisión general en las etapas del diseño conceptual y lógico. El diseño de su tabla debe ofrecer soporte para consultas en cada formulario y reporte. Las limitaciones en el diseño de la tabla aparecen a menudo como resultado de la implementación de formularios y reportes. Una vez implementados, se deben usar en cargas de trabajo típicas para asegurar un desempeño adecuado. Es probable que necesite ajustar el diseño físico para lograr este desempeño.

Esta sección demuestra la implementación de los requerimientos de datos para algunos formularios y reportes de la sección 13.1, además de un disparador para el mantenimiento de los datos derivados. La implementación de los requerimientos de datos para los otros formularios y reportes se dejaron como problemas para el final del capítulo.

##### *Requerimientos de datos para el formulario de origen de préstamo*

La lista siguiente proporciona las respuestas a los cinco pasos de requerimientos de datos que incluyen las consultas al formulario principal y el subformulario. Para propósitos de referencia, las figuras 13.2 y 13.3 muestran ejemplos del formulario de origen de datos.

- Identifique la relación 1-M manipulada por el formulario: la relación 1-M conecta la tabla *Loan* con la tabla *DisburseLine*.
- Identifique las columnas de enlace o unión para la relación 1-M: las columnas de unión son *Loan.LoanNo* y *DisburseLine.LoanNo*.

- Identifica las otras tablas en el formulario principal y en el subformulario: además de la tabla *Loan*, el formulario principal contiene las tablas *Student*, *Institution*, *Bank*, *Lender* y *Guarantor*. El subformulario no contiene tablas adicionales además de *DisburseLine*.
- Determina la capacidad para actualizar las tablas en forma jerárquica: la tabla *Loan* en el formulario principal y la tabla *DisburseLine* en el subformulario se pueden actualizar. Las otras tablas son de sólo lectura. Es necesario proporcionar formularios separados para actualizar las otras tablas que aparecen en el formulario principal.
- Escribe la consulta del formulario principal: El enlace exterior de un lado con la tabla *Bank* conserva la tabla *Loan*. El enlace exterior de un lado permite que los datos del banco aparezcan de manera opcional en el formulario. Los datos del banco aparecen en el formulario cuando el método de desembolso es electrónico. El enunciado SELECT recupera algunas columnas adicionales que no aparecen en el formulario, como *Bank.RouteNo*. Estas columnas adicionales no afectan la capacidad de actualización de la consulta.

```

SELECT Loan.*, Student.*, Institution.*, Bank.*, Institution.*,
       Lender.*
FROM ( ( ( Student INNER JOIN Loan ON Student.StdNo = Loan.StdNo )
           INNER JOIN Institution ON Institution.InstId = Loan.InstId
           INNER JOIN Lender ON Lender.LenderNo = Loan.LenderNo
           INNER JOIN Guarantor ON Guarantor.GuarantorNo = Loan.GuarantorNo
           LEFT JOIN Bank ON Bank.RouteNo = Loan.RouteNo
)

```

- Escriba la consulta al subformulario:

```

SELECT DisburseLine.*
FROM DisburseLine

```

#### *Requerimiento de datos para el reporte de actividad del préstamo*

La consulta al reporte es difícil de formular porque cada línea del Reporte de Actividad del Préstamo muestra los saldos inicial y final del préstamo. Es necesario usar dos filas en las tablas *Applied* y *Statement* para calcular los saldos inicial y final del préstamo. El saldo final se calcula como el valor de nota del préstamo menos el pago acumulado a la cantidad principal reflejado en la última fila *Applied* del año reportado. El saldo inicial del préstamo se calcula como el valor de nota del préstamo menos el pago acumulado a la cantidad principal en la última fila *Applied* del año anterior al reporte. Para determinar la última fila *Applied* para un año determinado, se enlaza la tabla *Applied* a la fila *Statement*, tomando la *DatePaid* más larga en el año. Las consultas empaquetadas en la cláusula WHERE recuperan las filas *Statement* con la máxima *DatePaid* para el año del reporte y el año anterior al reporte. El identificador *EnterReportYear* es el parámetro para el año del reporte. La función **Year** es una función de Microsoft Access que recupera la parte del año de una fecha.

```

SELECT Student.StdNo, Name, Address, Phone, City, State, Zip,
       Loan.LoanNo, NoteValue, ACurr.CumPrincipal, ACurr.CumInterest,
       APrec.CumPrincipal
FROM Student, Loan, Applied ACurr, Applied APrec,
       Statement SCurr, Statement SPrev
WHERE Student.StdNo = Loan.StdNo
      AND Loan.LoanNo = SCurr.LoanNo
      AND SCurr.StatementNo = ACurr.StatementNo
      AND ACurr.LoanNo = Loan.LoanNo
      AND SCurr.DatePaid =
        (   SELECT MAX(DatePaid) FROM Statement
            WHERE Year(DatePaid) = EnterReportYear  )

```

```

AND Loan.LoanNo = SPrev.LoanNo
AND SPrev.StatementNo = APrev.StatementNo
AND APrev.LoanNo = Loan.LoanNo
AND SPrev.DatePaid =
(   SELECT MAX(DatePaid) FROM Statement
    WHERE Year(DatePaid) = EnterReportYear - 1   )

```

Esta consulta de reporte demuestra la necesidad de las columnas calculadas *CumPrincipal* y *CumInterest*. Sería muy difícil formular la consulta de reportes sin estas columnas derivadas.

#### *Mantenimiento de datos derivados*

Se pueden definir los disparadores AFTER ROW para mantener las columnas derivadas en las tablas *Loan* y *Applied*. El siguiente código muestra un disparador de Oracle para mantener la columna *Loan.Balance* después de crear una fila *Applied*. Los disparadores para mantener las columnas *Applied.CumPrincipal* y *Applied.CumInterest* comprenden consideraciones de tablas mutantes en Oracle. Así como en el capítulo 11 no mostramos las soluciones para los disparadores con tablas mutantes, aquí tampoco mostraremos la solución para mantener estas columnas. La solución comprende un disparador INSTEAD OF con una vista o un paquete Oracle con un conjunto de disparadores.

```

CREATE OR REPLACE TRIGGER tr_Applied_IA
-- This trigger updates the Balance column
-- of the related Loan row.
AFTER INSERT
ON Applied
FOR EACH ROW
BEGIN
    UPDATE Loan
        SET Balance = Balance - :NEW.Principal
        WHERE LoanNo = :NEW.LoanNo;
EXCEPTION
    WHEN OTHERS THEN
        RAISE_Application_Error(-20001, 'Database error');
END;

```

---

## Reflexión final

Este capítulo presentó el estudio de un caso de tamaño moderado como muestra del proceso de desarrollo de bases de datos. El caso de Student Loan Limited describió un importante subconjunto del procesamiento de préstamos comerciales a los estudiantes que incluye préstamos aceptados de prestamistas, notificaciones de pago a los estudiantes, facturación y procesamiento de pagos, y reportes sobre las condiciones de los préstamos. La solución del caso integró las técnicas presentadas en los capítulos de las partes 2 a 5. La solución ilustró los modelos y la documentación producida en el modelado conceptual, el diseño lógico de bases de datos y las etapas de diseño físico de bases de datos, así como requerimientos de datos para formularios, reportes y disparadores.

Después de leer con detenimiento este capítulo, está preparado para manejar el desarrollo de bases de datos para una organización real. Lo alentamos a que trabaje en los casos disponibles en el sitio web del libro para reforzar su comprensión del proceso de desarrollo de bases de datos. Este caso, aunque presenta un problema mayor y más integrado que los otros capítulos, no es comparable a llevar a cabo el desarrollo de bases de datos para una organización real. Para las organizaciones los requerimientos a menudo son ambiguos, incompletos e inconsistentes. Para el éxito a largo plazo, resulta crucial decidir los límites de las bases de datos y modificar

el diseño de las bases en respuesta a los cambios en los requerimientos. El hecho de vigilar la operación de la base de datos le permite mejorar el desempeño de acuerdo con su uso. Estos retos hacen que el desarrollo de bases de datos sea una actividad que estimula el intelecto.

## Revisión de conceptos

- El programa Guaranteed Student Loan (GSL) ofrece préstamos subsidiados y no subsidiados.
- La función de los estudiantes, prestamistas, proveedores de servicios, fiadores y reguladores gubernamentales en el programa GSL.
- Flujo de trabajo para el procesamiento de los préstamos de los estudiantes, que comprende aplicaciones de préstamos, aprobación de préstamos, origen de préstamos, separación de la escuela y pago de préstamos.
- Documentos importantes para procesar préstamos: el formulario de origen del préstamo, la carta de declaración, el estado de cuenta y el reporte de actividad del préstamo.
- Modelado de datos conceptuales: estrategia de integración incremental para el formulario de origen del préstamo, la carta de declaración, el estado de cuenta y el reporte de actividad del préstamo.
- Conversión del ERD utilizando las reglas de conversión básicas.
- Eliminar las violaciones a los formularios normales en las tablas *Loan*, *Student* e *Institution*.
- Especificación de los perfiles de tablas y aplicaciones para el diseño físico de bases de datos.
- Aplicar las reglas de selección de índices para los índices agrupados sobre las llaves primarias y los índices no agrupados sobre las llaves foráneas.
- Uso de la desnormalización para la tabla *Loan*.
- Especificar los requerimientos de datos para el formulario de origen del préstamo y el reporte de actividad del préstamo, con el fin de llevar a cabo una revisión general del resultado de las etapas de modelado conceptual y diseño lógico de datos.
- Escribir disparadores para mantener los datos derivados en las tablas *Loan* y *Applied*.

## Preguntas

1. ¿Por qué el proceso de solicitudes por parte de los estudiantes no se considera en la etapa del diseño conceptual?
2. ¿Por qué se utiliza el planteamiento de integración incremental para analizar los requerimientos?
3. ¿Por qué se analiza primero el formulario de origen del préstamo?
4. ¿Cómo se relaciona el campo de valor de notas en el formulario de origen del préstamo con los otros datos en el formulario?
5. Explique cómo se representa la relación 1-M del formulario de origen del préstamo en el ERD de la figura 13.9.
6. ¿Cuál es la llave primaria del tipo de entidad *DisburseLine* en la figura 13.9?
7. ¿Qué datos contiene el atributo de imagen del tipo de entidad *DiscLetter* en la figura 13.11?
8. Explique cómo se representa la relación 1-M del estado de cuenta en el ERD de la figura 13.13.
9. ¿Por qué la regla de la relación 1-M opcional (regla 5 del capítulo 9) no se utiliza para convertir el ERD de la figura 13.15?
10. Explique de qué manera la relación *Authorizes* de la figura 13.15 se convierte en la figura 13.16.
11. Explique de qué manera la dependencia de identificación de la figura 13.15 se convierte en la figura 13.16.
12. Explique de qué manera la relación *Applied* de la figura 13.15 se convierte en la figura 13.16.
13. Explique por qué la tabla *DiscLetter* está en BCNF.

14. Analice una posible justificación para la violación de BCNF con la tabla *Student* que ilustra la tabla 13.7.
15. ¿Por qué dividir los documentos en varias aplicaciones de bases de datos como se ilustra en la tabla 13.8?
16. Explique la diferencia entre el procesamiento en lote y en línea de los formularios de origen del préstamo. ¿Por qué el procesamiento en lote es factible para los formularios de origen del préstamo?
17. ¿De qué manera el procesamiento en lote puede reducir el impacto del mantenimiento de los índices?
18. Explique por qué se recomienda un índice agrupado para la columna *Applied.StatementNo*.
19. Explique por qué se recomienda un índice no agrupado para la columna *Applied.LoanNo*.
20. Explique la relación entre la columna *Loan.NoteValue* y las columnas *Amount*, *OrigFee* y *GuarFee* en la tabla *DisburseLine*.

## Problemas

Los siguientes problemas comprenden extensiones al caso de Student Loan Limited. Para casos adicionales de complejidad similar, visite el sitio web de este libro.

1. Emplee la regla de la relación 1-M opcional para convertir la relación *Guarantees* en la figura 13.15. Modifique el diagrama del modelo relacional de la figura 13.16 con el cambio de conversión.
2. Simplifique el ERD para el formulario de origen del préstamo (figura 13.9) combinando el tipo de entidad *Loan* con los tipos de entidad asociados con un préstamo (*Lender* y *Guarantor*). ¿Qué transformación se utiliza para combinar los tipos de entidad? (vea el capítulo 6) ¿Qué transformación se puede usar para dividir los atributos del banco (*RouteNo* y *DisbBank*) en un tipo de entidad separado?
3. Modifique el ERD en la figura 13.15 para reflejar un cambio en la relación entre un reporte de actividad y los préstamos asociados de un estudiante. La suposición en este caso es que un reporte de actividad resume todos los préstamos de un estudiante. La nueva suposición es que un reporte de actividad podría resumir sólo un subconjunto de préstamos de un estudiante.
4. Explique de qué manera se puede usar la desnormalización para combinar las tablas *LoanActivity* y *Student*. Haga lo mismo para las tablas *DiscLetter* y *Loan*.
5. Student Loan Limited decidió entrar en el negocio de los préstamos directos. Un préstamo directo es similar a un préstamo para estudiantes con fiador, sólo que en el primero no hay prestamista ni fiador. Debido a la falta de éstos, tampoco hay cuotas de origen ni fianza. Sin embargo, hay una cuota de servicio de casi 3 por ciento del valor de la nota. Además, un estudiante podría elegir la devolución de ingresos-contingentes después de separarse de la escuela. Si un estudiante elige este tipo de pago, se revisan los términos del préstamo y la cantidad del pago.
  - a) Modifique el ERD de la figura 13.15 para que refleje estos nuevos requerimientos.
  - b) Convierta los cambios del ERD en el diseño de una tabla. Muestre el resultado de la conversión como una modificación al diagrama de la base de datos relacional en la figura 13.16.
6. Student Loan Limited no puede justificar los gastos en software y hardware de imágenes. Por tanto, es preciso modificar el diseño de la base de datos. No es posible guardar las columnas *Image* en las tablas *DiscLetter* y *LoanActivity*. En vez de ello, los datos guardados en las columnas de imagen se deben guardar o calcular de acuerdo con la demanda.
  - a) Haga las recomendaciones pertinentes para guardar o calcular los campos subrayados en una carta de declaración. Modifique el diseño de la tabla según sea necesario. Considere entre sus recomendaciones cambios de actualización y recuperación.
  - b) Haga las recomendaciones pertinentes para guardar o calcular los campos subrayados en un reporte de actividad del préstamo. Considere entre sus recomendaciones cambios de actualización y recuperación.
7. Escriba un enunciado SELECT para indicar los requerimientos de datos para la carta de declaración que ilustra la figura 13.4.
8. Use los cinco pasos que se presentan en el capítulo 10 para especificar los requerimientos de datos para el formulario estado de cuenta que ilustra la figura 13.5.
9. ¿Qué aspectos comprende la implementación de la relación entre la columna *Loan.NoteValue* y las columnas *Amount*, *OrigFee* y *GuarFee* en la tabla *DisburseLine*?
10. ¿Por qué un disparador Oracle para mantener las columnas *Applied.CumPrincipal* y *Applied.CumInterest* implica consideraciones de tablas mutantes?

## Apéndice 13.A

### Glosario de los campos del formulario y reporte

El apéndice 13.A ofrece una breve descripción de los campos que se encuentran en los documentos que presentamos en la sección 13.1. Los nombres de los campos son los títulos del documento asociado.

#### Loan Origination Form

- *Loan No.:* valor alfanumérico único que identifica un formulario de origen del préstamo.
- *Date:* fecha en que se llenó el formulario de origen del préstamo.
- *Student No.:* valor alfanumérico único que identifica al estudiante.
- *Name:* nombre del estudiante que solicita el préstamo.
- *Address:* dirección física del estudiante que solicita el préstamo.
- *City, State, Zip:* concatenación de la ciudad, el estado y el código postal del estudiante.
- *Phone:* número de teléfono del estudiante que incluye el código de área.
- *Date of Birth:* fecha de nacimiento del estudiante que solicita el préstamo.
- *Expected Graduation:* mes y año esperado para la graduación.
- *Institution ID:* número de identificación federal de la universidad o escuela.
- *Institution Name:* nombre de la universidad o escuela.
- *Address:* dirección física de la universidad o escuela.
- *City, State, Zip:* concatenación de la ciudad, el estado y el código postal de la institución.
- *Disbursement Method:* el método utilizado para distribuir los fondos para el estudiante que solicita el préstamo; los valores pueden ser EFT (transferencia electrónica de fondos) o cheque.
- *Routing No.:* valor alfanumérico único que identifica un banco para el desembolso de fondos; sólo se usa si el método de desembolso es EFT.
- *Account No.:* valor alfanumérico único que identifica una cuenta del estudiante que solicita el préstamo; Account No. sólo garantiza ser único en el banco del estudiante (identificado por el número de ruta).
- *Disbursement Bank:* nombre del banco del que se desembolsan los fondos; sólo se utiliza si el método de desembolso es EFT.
- *Lender No.:* valor alfanumérico único que identifica la institución financiera que presta los fondos al estudiante que solicita el préstamo.
- *Lender Name:* nombre de la institución financiera que hace el préstamo al estudiante que lo solicita.
- *Guarantor No.:* valor alfanumérico único que identifica a la institución financiera que garantiza el pago apropiado del préstamo.
- *Guarantor Name:* nombre de la institución financiera que otorga la fianza.
- *Note Value:* cantidad (en dólares) que el estudiante pide prestado; el valor de nota es igual a la suma de las cantidades del desembolso y las cuotas (de origen y fianza).
- *Subsidized:* valor si/no que indica si el gobierno paga el interés mientras el estudiante está en la escuela.
- *Rate:* tasa de interés sobre el préstamo.
- *Date:* fecha de desembolso: éste es el campo Date en Disbursement Plan.
- *Amount:* cantidad del desembolso en dólares.
- *Origination Fee:* cuota (en dólares) que cobra la institución prestamista.
- *Guarantee Fee:* cuota (en dólares) que cobra el fiador.

## Disclosure Letter

- *Date*: fecha (1 Julio 2005) en que la carta se envió al estudiante que solicita el préstamo.
- *Loan No.*: número asociado del préstamo.
- *Last Name*: título y apellido (señor estudiante) del estudiante que solicita el préstamo.
- *Repayment Starting*: mes y año (Septiembre 2005) en que los préstamos entran en estatus de pago.
- *Amount Borrowed*: suma de las cantidades solicitadas (\$10 000) en todos los préstamos que cubre el plan de pagos.
- *Number of Payments*: cantidad estimada de pagos programados (120) para cubrir la cantidad solicitada.
- *Interest Rate*: tasa porcentual promedio ponderada (8.5 por ciento) de los préstamos que cubre el plan de pagos.
- *Finance Charge*: cargo financiero estimado (4 877.96 dólares) si la cantidad solicitada se paga de acuerdo con el plan de pagos.
- *Payment Amount*: cantidad del pago requerido para cada mes (246.37 dólares, excepto quizás para el último mes). Si un estudiante no paga esta cantidad cada mes, se declarará moroso, a menos que se llegue a un acuerdo.
- *First Payment Date*: fecha en que se tiene que cubrir el primer pago (Octubre 31, 2005), en caso de seguir el plan de pagos.
- *Last Payment Date*: fecha en que se debe hacer el último pago (Septiembre 30, 2015), en caso de seguir el plan de pagos.

## Statement of Account

- *Statement No.*: valor alfanumérico único (B100) que identifica el formulario del estado de cuenta.
- *Date*: fecha en que se envió el estado de cuenta.
- *Student No.*: valor alfanumérico único que identifica a un estudiante.
- *Name*: nombre del estudiante que solicita el préstamo.
- *Address*: dirección física del estudiante que solicita el préstamo (parte de la dirección de correo postal).
- *City*: ciudad del estudiante que solicita el préstamo (parte de la dirección de correo postal).
- *State*: abreviatura en dos letras del nombre del estado del estudiante que solicita el préstamo (parte de la dirección de correo postal).
- *Zip*: código postal a cinco o nueve dígitos del estudiante que solicita el préstamo (parte de la dirección de correo postal).
- *Amount Due*: cantidad (en dólares) que el estudiante debe cubrir.
- *Due Date*: fecha en que Student Loan Limited debe recibir el pago. Es probable que se evalúe una multa si la cantidad se recibe en una fecha posterior.
- *Payment Method*: forma de pago (cheque o EFT).
- *Amount Enclosed*: cantidad (en dólares) enviada con el pago. Si el método de pago es EFT, el solicitante no tiene que llenar este campo.
- *Loan No.*: valor alfanumérico único que identifica un préstamo del solicitante.
- *Balance*: saldo pendiente del préstamo (en dólares) antes del pago.
- *Rate*: tasa de interés porcentual que se aplica al préstamo.
- *Date Paid*: fecha en que se recibió el pago; el personal de Student Loan Limited debe llenar este campo.

## Reporte de actividad del préstamo

- *Date*: fecha en que se elaboró el reporte.
- *Student No.*: valor alfanumérico único que identifica a un estudiante.
- *Name*: nombre del estudiante que solicita el préstamo.
- *Street*: dirección física del estudiante que solicita el préstamo (parte de la dirección de correo postal).
- *City*: ciudad del estudiante que solicita el préstamo (parte de la dirección de correo postal).
- *State*: abreviatura en dos letras del estado del estudiante que solicita el préstamo (parte de la dirección de correo postal).
- *Zip*: código postal a cinco o nueve dígitos del estudiante que solicita el préstamo (parte de la dirección de correo postal).
- *Loan No.*: valor alfanumérico único que identifica un préstamo del solicitante.
- *Beg. Balance*: saldo pendiente del préstamo (en dólares) al principio del año.
- *Principal*: cantidad total de los pagos aplicados a la cantidad principal.
- *Interest*: monto total de los pagos aplicados al interés.
- *Ending Balance*: saldo pendiente del préstamo (en dólares) al final del año después de aplicar los pagos.

## Apéndice 13.B

### Enunciados CREATE TABLE

El apéndice 13.B contiene enunciados CREATE TABLE para las tablas que resultan del proceso de conversión y normalización que describimos en la sección 13.3. Los enunciados CREATE TABLE cumplen con la sintaxis SQL:2003.

```
CREATE TABLE Student
(
    StdNo      CHAR(10),
    Name       CHAR(30)   CONSTRAINT StdNameRequired NOT NULL,
    Address    VARCHAR(50) CONSTRAINT StdAddressRequired NOT NULL,
    Phone      CHAR(9),
    City       CHAR(30)   CONSTRAINT StdCityRequired NOT NULL,
    Zip        CHAR(9)   CONSTRAINT StdZipRequired NOT NULL,
    ExpGradMonth SMALLINT,
    ExpGradYear INTEGER,
    DOB        DATE      CONSTRAINT StdDOBRequired NOT NULL,
    CONSTRAINT FKZip1 FOREIGN KEY (Zip) REFERENCES ZipCode
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
    CONSTRAINT PKStudent PRIMARY KEY (StdNo) )
```

```
CREATE TABLE Lender
(
    LenderNo    INTEGER,
    Name        CHAR(30)  CONSTRAINT LendNameRequired NOT NULL,
    CONSTRAINT PKLender PRIMARY KEY (LenderNo) )
```

```
CREATE TABLE Guarantor
  (  GuarantorNo      CHAR(10),
     Name            CHAR(30) CONSTRAINT GrnNameRequired NOT NULL,
   CONSTRAINT PKGuarantor PRIMARY KEY (GuarantorNo)  )
```

```
CREATE TABLE Institution
  (  InstID      CHAR(10),
     Name        CHAR(30)  CONSTRAINT InstNameRequired NOT NULL,
     Address    VARCHAR(50) CONSTRAINT InstAddressRequired NOT NULL,
     City       CHAR(30)   CONSTRAINT InstCityRequired NOT NULL,
     Zip        CHAR(9)    CONSTRAINT InstZipRequired NOT NULL,
   CONSTRAINT FKZip2 FOREIGN KEY (Zip) REFERENCES ZipCode
     ON DELETE RESTRICT
     ON UPDATE CASCADE,
   CONSTRAINT PKInstitution PRIMARY KEY (InstID)  )
```

```
CREATE TABLE ZipCode
  (  Zip          CHAR(9),
     State        CHAR(2)  CONSTRAINT ZipStateRequired NOT NULL,
   CONSTRAINT PKZipCode PRIMARY KEY (Zip)  )
```

```
CREATE TABLE Loan
  (  LoanNo      CHAR(10),
     ProcDate    DATE     CONSTRAINT LoanProcDateRequired NOT NULL,
     DisbMethod  CHAR(6)   CONSTRAINT LoanDisbMethodRequired NOT NULL,
     RouteNo    CHAR(10),
     AcctNo     CHAR(10),
     DateAuth    INTEGER  CONSTRAINT LoanDateAuthRequired NOT NULL,
     NoteValue   DECIMAL(10,2) CONSTRAINT LoanNoteValueRequired NOT NULL,
     Subsidized  BOOLEAN  CONSTRAINT LoanSubsidizedRequired NOT NULL,
     Rate        FLOAT    CONSTRAINT LoanRateRequired NOT NULL,
     Balance    DECIMAL(10,2),
     StdNo      CHAR(10)  CONSTRAINT LoanStdNoRequired NOT NULL,
     InstID     CHAR(10)  CONSTRAINT LoanInstIdRequired NOT NULL,
     GuarantorNo CHAR(10),
     LenderNo   CHAR(10)  CONSTRAINT LoanLenderNoRequired NOT NULL,
   CONSTRAINT FKStdNo1 FOREIGN KEY (StdNo) REFERENCES Student
     ON DELETE RESTRICT
     ON UPDATE CASCADE,
   CONSTRAINT FKInstID FOREIGN KEY (InstID) REFERENCES Institution
     ON DELETE RESTRICT
     ON UPDATE CASCADE,
   CONSTRAINT FKGuarantorNo FOREIGN KEY (GuarantorNo) REFERENCES Guarantor
     ON DELETE RESTRICT
     ON UPDATE CASCADE,
```

```

CONSTRAINT FKLenderNo FOREIGN KEY (LenderNo) REFERENCES Lender
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
CONSTRAINT FKRouteNo FOREIGN KEY (RouteNo) REFERENCES Bank
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
CONSTRAINT PKLoan PRIMARY KEY (LoanNo)  )

```

```

CREATE TABLE Bank
(
    RouteNo      CHAR(10),
    Name         CHAR(30) CONSTRAINT BankNameRequired NOT NULL,
CONSTRAINT PKBank PRIMARY KEY (RouteNo)  )

```

```

CREATE TABLE DisburseLine
(
    LoanNo       CHAR(10),
    DateSent    DATE,
    Amount       DECIMAL(10,2) CONSTRAINT DLAmountRequired NOT NULL,
    OrigFee     DECIMAL(10,2) CONSTRAINT DLOrigFeeRequired NOT NULL,
    GuarFee     DECIMAL(10,2) CONSTRAINT DLGuarFeeRequired NOT NULL,
CONSTRAINT FKLoanNo1 FOREIGN KEY (LoanNo) REFERENCES Loan
    ON DELETE CASCADE
    ON UPDATE CASCADE,
CONSTRAINT PKDisburseLine PRIMARY KEY (LoanNo, DateSent)  )

```

```

CREATE TABLE DiscLetter
(
    LetterNo    INTEGER,
    DateSent    DATE    CONSTRAINT DLDatesentRequired NOT NULL,
    Image       BLOB    CONSTRAINT DLImageRequired NOT NULL,
    LoanNo      CHAR(10) CONSTRAINT DLLoanNoRequired NOT NULL,
CONSTRAINT FKLoanNo2 FOREIGN KEY (LoanNo) REFERENCES Loan
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
CONSTRAINT PKDiscLetter PRIMARY KEY (LetterNo)  )

```

```

CREATE TABLE LoanActivity
(
    ReportNo   INTEGER,
    DateSent   DATE    CONSTRAINT LADateSentRequired NOT NULL,
    Image      BLOB    CONSTRAINT LAImageRequired NOT NULL,
    StdNo      CHAR(10) CONSTRAINT LAStdNoRequired NOT NULL,

```

```

CONSTRAINT FKStdNo2 FOREIGN KEY (StdNo) REFERENCES Student
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
CONSTRAINT PKLoanActivity PRIMARY KEY (ReportNo)  )

```

## CREATE TABLE Statement

```

( StatementNo   CHAR(10),
  StatementDate DATE      CONSTRAINT StmtDateRequired NOT NULL,
  PayMethod     CHAR(6)   CONSTRAINT StmtPayMethodRequired NOT NULL,
  StdNo         CHAR(10)  CONSTRAINT StmtStdNoRequired NOT NULL,
  AmtDue        DECIMAL(10,2) CONSTRAINT StmtAmtDuetRequired NOT NULL,
  DueDate       DATE      CONSTRAINT StmtDueDateRequired NOT NULL,
  AmtSent       DECIMAL(10,2),
  DatePaid      DATE,
CONSTRAINT FKStdNo3 FOREIGN KEY (StdNo) REFERENCES Student
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
CONSTRAINT PKStatement PRIMARY KEY (StatementNo)  )

```

## CREATE TABLE Applied

```

( LoanNo        CHAR(10),
  StatementNo   CHAR(10),
  Principal     DECIMAL(10,2) CONSTRAINT AppPrincipal NOT NULL,
  Interest      DECIMAL(10,2) CONSTRAINT ApplInterest NOT NULL,
  CumPrincipal  DECIMAL(10,2) CONSTRAINT AppCumPrincipal NOT NULL,
  CumInterest   DECIMAL(10,2) CONSTRAINT AppCumInterest NOT NULL,
CONSTRAINT FKLoanNo3 FOREIGN KEY (LoanNo) REFERENCES Loan
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
CONSTRAINT FKStatementNo FOREIGN KEY (StatementNo) REFERENCES Statement
    ON DELETE CASCADE
    ON UPDATE CASCADE,
CONSTRAINT PKApplied PRIMARY KEY (LoanNo, StatementNo)  )

```



Parte

7

# Administración de entornos de bases de datos

---

Los capítulos de la parte 7 enfatizan la función del especialista en base de datos y los detalles para administrar las bases en diversos entornos operativos. El capítulo 14 ofrece un contexto para los demás, al cubrir las responsabilidades, herramientas y procesos que emplean los administradores de bases de datos y los administradores de datos. Los otros capítulos de esta parte proporcionan los fundamentos para administrar bases de datos en entornos importantes: el capítulo 15 en el procesamiento de transacciones, el capítulo 16 en *data warehouses*, el capítulo 17 en procesamiento distribuido, bases de datos paralelas y datos distribuidos, y el capítulo 18 en administración de bases de datos de objetos. Estos capítulos enfatizan conceptos, arquitecturas e importantes decisiones de diseño para los especialistas en bases de datos. Además, los capítulos 15, 16 y 18 proporcionan detalles sobre las sentencias SQL que se utilizan en el procesamiento de transacciones, el desarrollo de *data warehouses* y el desarrollo de bases de datos de objetos.

---

**Capítulo 14.** Administración de datos y bases de datos

**Capítulo 15.** Administración de transacciones

**Capítulo 16.** Tecnología y administración de data warehouse

**Capítulo 17.** Procesamiento cliente-servidor, procesamiento de bases de datos paralelas y bases de datos distribuidas

**Capítulo 18.** Sistemas de administración de bases de datos de objetos



## Capítulo

# 14

# Administración de datos y bases de datos

### Objetivos de aprendizaje

Este capítulo ofrece un panorama general de las responsabilidades y herramientas de los especialistas en bases de datos, conocidos como administradores de datos y administradores de bases de datos. Al finalizar este capítulo, el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Comparar y contrastar las responsabilidades de los administradores de bases de datos y los administradores de datos.
- Controlar bases de datos utilizando sentencias SQL para seguridad e integridad.
- Administrar procedimientos almacenados y disparadores (*triggers*).
- Entender las funciones de las tablas de diccionario de datos y el diccionario de recursos de información.
- Describir el proceso de planeación de datos.
- Entender el proceso para seleccionar y evaluar DBMS.
- Conocer los entornos de procesamiento en los que se usa la tecnología de bases de datos.

### Panorama general

Con la aplicación de los conocimientos y las habilidades de las partes 1 a 6, podrá desarrollar bases de datos e implementar aplicaciones que las utilicen. Ya aprendió el modelado conceptual de datos, el diseño de bases de datos relacional, la formulación de consultas, el desarrollo de aplicaciones con vistas, los procedimientos almacenados, los disparadores y el desarrollo de bases de datos utilizando los requerimientos representados como vistas. La parte 7 complementa estas áreas de conocimientos y habilidades estudiando los aspectos y las habilidades relacionados con la administración de bases de datos en distintos entornos de procesamiento. Este capítulo describe las responsabilidades y las herramientas de los especialistas en datos (administradores de datos y de bases de datos) y proporciona una introducción a los distintos entornos de procesamiento para bases de datos.

Antes de estudiar los detalles de los entornos de procesamiento, debe entender el contexto organizacional en el que existen las bases de datos y aprender las herramientas y los procesos para administrarlas. Este capítulo analiza primero un contexto organizacional para las bases de datos. Aprenderá sobre el apoyo que ofrecen las bases de datos a la toma de decisiones administrativas,

las metas de la administración de recursos de información y las responsabilidades de los administradores de datos y bases de datos. Después de explicar el contexto organizacional, este capítulo presenta nuevas herramientas y procesos para administrar bases de datos. Va a aprender sentencias SQL para seguridad e integridad, el manejo de disparadores y procedimientos almacenados y la manipulación de diccionarios de datos, así como los procesos para la planeación de datos y la selección de DBMS. Este capítulo termina con una introducción a los distintos entornos de procesamiento que presentaremos con mayor detalle en los otros capítulos de la parte 7.

## 14.1 Contexto organizacional para administrar bases de datos

Esta sección revisa los niveles de toma de decisión administrativa y analiza el apoyo que ofrecen las bases de datos en la toma de decisiones a todos los niveles. Después de estos antecedentes, describe la función de la administración de recursos de información y las responsabilidades de los especialistas en datos al manejar estos recursos.

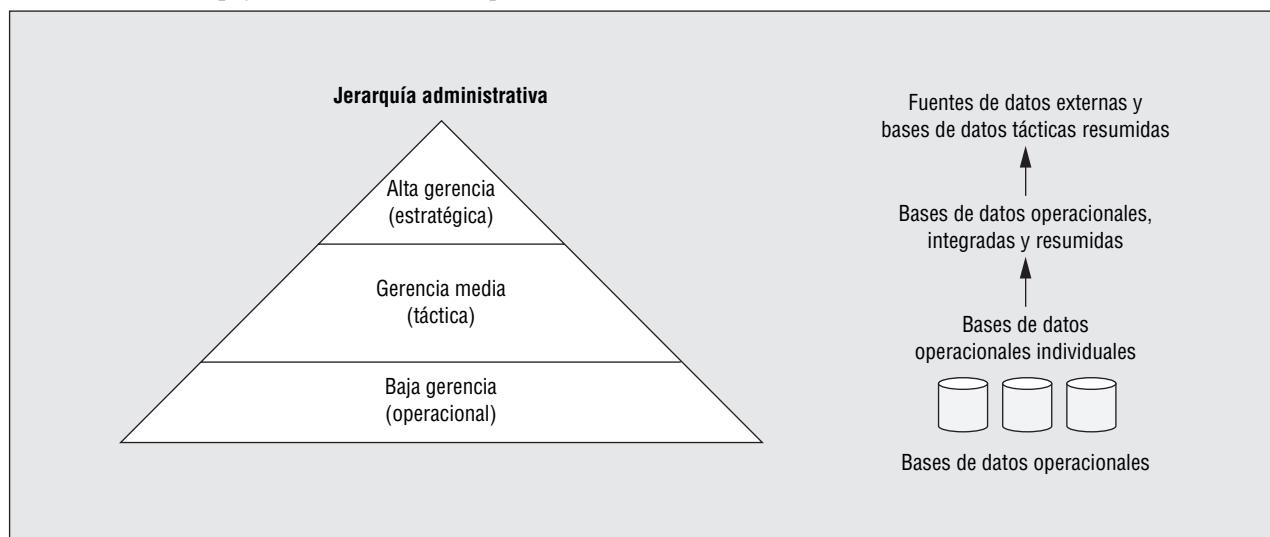
### 14.1.1 Apoyo de las bases de datos en la toma de decisiones administrativas

**base de datos operacional**  
una base de datos que soporta las funciones cotidianas de una organización.

Las bases de datos soportan las operaciones de negocios y la toma de decisiones administrativas en distintos niveles. La mayoría de las grandes organizaciones han desarrollado diversas bases de datos operacionales para ayudar a manejar los negocios con eficiencia. Las bases de datos operacionales soportan en forma directa las principales funciones, como procesamiento de pedidos, manufactura, cuentas por pagar y distribución de productos. Las razones para invertir en una base de datos operacional casi siempre son un rápido procesamiento, mayores volúmenes de negocios y reducidos costos de personal.

Conforme las organizaciones logran mejorar su operación, empiezan a darse cuenta del potencial de toma de decisiones de sus bases de datos. Las bases de datos operacionales ofrecen la materia prima para la toma de decisiones administrativas, como ilustra la figura 14.1. La gerencia de bajo nivel puede obtener reportes de excepciones y problemas directamente de bases de datos operacionales. Sin embargo, es preciso agregar mayor valor con el fin de aprovechar las bases de datos operacionales para la gerencia media y alta. Es necesario limpiar, integrar y resumir las bases de datos operacionales con el objeto de proporcionar valor para la toma de decisiones tácticas y estratégicas. La integración se vuelve necesaria porque a menudo las bases

**FIGURA 14.1** Apoyo de las bases de datos para los niveles administrativos



de datos operacionales se desarrollan en forma aislada, sin considerar las necesidades de información para la toma de decisiones tácticas y estratégicas.

La tabla 14.1 ofrece ejemplos de decisiones administrativas y requerimientos de datos. La baja gerencia se ocupa de problemas a corto plazo relacionados con transacciones individuales. Los resúmenes periódicos de bases de datos operacionales y los reportes de excepciones ayudan a la gerencia operativa. Tal vez la gerencia media quiera integrar los datos en distintos departamentos, plantas de manufactura y tiendas detallistas. La alta gerencia depende de los resultados del análisis de la gerencia media y las fuentes de datos externas. La alta gerencia necesita integrar los datos de modo que sea posible llevar un registro de clientes, productos, proveedores y otras entidades importantes en toda la organización. Además, es necesario resumir los datos externos y luego integrarlos con los internos.

### 14.1.2 Administración de los recursos de información para el manejo del conocimiento

#### ciclo de vida de la información

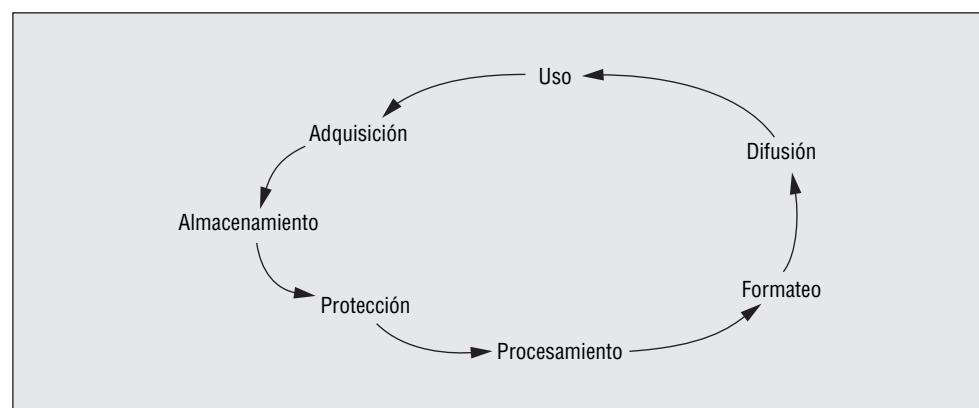
las etapas de transformación de la información en una organización. Cada entidad tiene su propio ciclo de vida de la información que es necesario manejar e integrar con los ciclos de vida de otras entidades.

La filosofía de la administración de recursos de información surge como respuesta a los desafíos del aprovechamiento de las bases de datos operacionales y la tecnología de la información para la toma de decisiones administrativas, que comprende procesamiento, distribución e integración de la información en toda la organización. Un elemento clave de la administración de recursos de información es el control de los ciclos de vida de la información (figura 14.2). Cada nivel de toma de decisiones administrativas y operaciones de negocios tiene un ciclo de vida de la información propio. Para una efectiva toma de decisiones, es necesario integrar los ciclos de vida para proporcionar información oportuna y consistente. Por ejemplo, los ciclos de vida de la información para las operaciones ofrecen datos para la toma de decisiones administrativas.

**TABLA 14.1**  
Ejemplos de toma de decisiones gerenciales

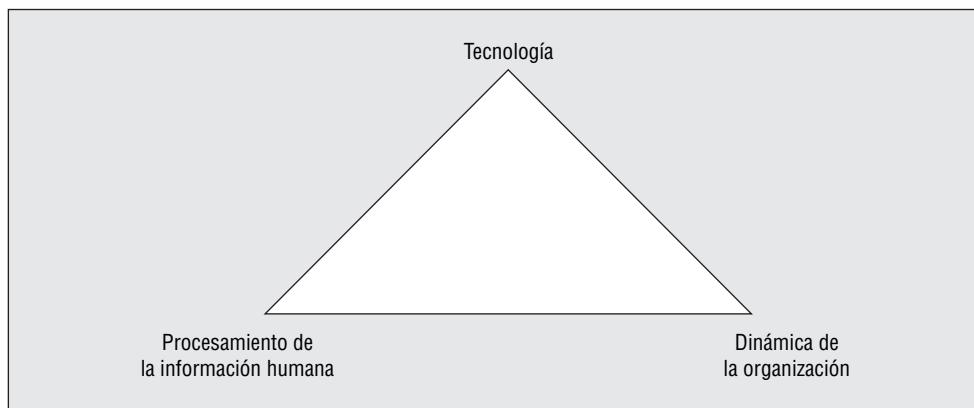
Nivel	Ejemplo de decisiones	Requerimientos de datos
Alto	Identificar nuevos mercados y productos; planeación del crecimiento; redistribuir los recursos entre las divisiones.	Proyecciones económicas y tecnológicas; resúmenes de noticias; reportes de la industria; reportes del desempeño a mediano plazo.
Medio	Elegir los proveedores; proyección de ventas, inventario y efectivo; revisar los niveles de personal; preparar presupuestos.	Tendencias históricas; desempeño de proveedores; análisis de caminos críticos; planes a corto y mediano plazo.
Bajo	Programar los horarios de los empleados; corregir las demoras en los contratos; detectar cuellos de botella en la producción; monitoreo del uso de recursos.	Reportes de problemas; reportes de excepciones; horarios de empleados; resultados de la producción diaria; niveles de inventario.

**FIGURA 14.2**  
Etapas típicas de un ciclo de vida de la información



**FIGURA 14.3**

Tres pilares del manejo del conocimiento



La calidad de los datos es una preocupación importante para la administración de los recursos de información por el impacto que tiene en la toma de decisiones administrativas. Como estudiamos en el capítulo 2, la calidad de los datos comprende varias dimensiones, como corrección, oportunidad, consistencia y confiabilidad. A menudo, el nivel de calidad de los datos que es suficiente para las operaciones de negocios puede no serlo para la toma de decisiones en niveles gerenciales más altos. Este conflicto se presenta sobre todo en la dimensión de la consistencia. Por ejemplo, la inconsistencia en la identificación de los clientes en las bases de datos operativas puede afectar la toma de decisiones en el nivel de la alta gerencia. La administración de los recursos de información enfatiza una perspectiva general de la organización a largo plazo sobre la calidad de los datos para asegurar el apoyo en la toma de decisiones administrativas.

### **administración del conocimiento**

aplicación de la tecnología de la información con las capacidades de procesamiento de información humana y los procesos de la organización para apoyar una rápida adaptación al cambio.

En años recientes, surgió un movimiento para ampliar la administración de recursos de información hacia la administración de la información. Por tradición, la administración de recursos de información ha enfatizado la tecnología en apoyo de las fórmulas predefinidas para la toma de decisiones, en lugar de la capacidad de reaccionar a un entorno de negocios en constante cambio. Para tener éxito en el entorno de negocios actual, las organizaciones deben poner énfasis en una respuesta rápida y adaptarse, en lugar de realizar planeación. Para enfrentar este reto, el doctor Yogesh Malhotra, un conocido asesor en administración, argumenta que las organizaciones deben desarrollar sistemas que faciliten la creación de conocimientos en lugar de la administración de la información. Para la creación de conocimientos sugiere mayor hincapié en el procesamiento humano de la información y la dinámica de la organización para equilibrar el énfasis en la tecnología, como muestra la figura 14.3.

Esta visión de la administración del conocimiento ofrece un contexto para el uso de la tecnología de la información en la solución de problemas de negocios. La mejor tecnología de la información fracasa si no se alinea con los elementos humanos y de la organización. La tecnología de la información debe ampliar la capacidad intelectual individual, compensar las limitaciones en el procesamiento humano y apoyar una dinámica positiva en la organización.

#### **14.1.3 Responsabilidades de los administradores de datos y los administradores de bases de datos**

Como parte del control de recursos de información, han surgido nuevas responsabilidades para la administración. El administrador de datos (DA) es un puesto de gerencia alta o media con amplias responsabilidades en cuanto a la administración de los recursos de información. El administrador de bases de datos (DBA) es una función de apoyo con responsabilidades relacionadas con las bases de datos individuales y los DBMS. La tabla 14.2 compara las responsabilidades de los administradores de datos y los administradores de bases de datos. El administrador de datos ve el recurso de la información en un contexto más amplio que el administrador de bases de datos. El primero considera todo tipo de datos, sin importar si están almacenados en bases de datos de relación, archivos, páginas web o fuentes externas. Por lo general, el administrador de bases de datos sólo considera la información almacenada en bases de datos.

**TABLA 14.2**  
**Responsabilidades de los administradores de datos y los administradores de bases de datos**

Puesto	Responsabilidades
Administrador de datos	Desarrolla un modelo empresarial de datos. Establece estándares y políticas entre las bases de datos con respecto a los nombres, la capacidad para compartir datos y la propiedad de los datos. Negocia los términos contractuales con los distribuidores de tecnología de la información.
Administrador de bases de datos	Desarrolla planes a largo plazo para la tecnología de la información. Desarrolla el conocimiento detallado sobre los DBMS individuales. Realiza consultas sobre el desarrollo de aplicaciones. Lleva a cabo el modelado de datos y el diseño lógico de bases de datos. Pone en práctica los estándares de administración de datos. Vigila el desempeño de las bases de datos. Lleva a cabo la evaluación técnica de los DBMS. Crea sentencias de seguridad, integridad y procesamiento de reglas. Desarrolla estándares y políticas relacionadas con bases de datos individuales y DBMS.

### **modelo de datos empresariales**

un modelo de datos conceptual en una organización. Un modelo de datos empresariales se puede usar para la planeación de los datos y para apoyar las decisiones.

Una de las responsabilidades más importantes del administrador de datos es el desarrollo de un modelo de datos empresariales. El modelo de datos empresariales proporciona un patrón integrado de todas las bases de datos de una organización. Debido a su alcance, un modelo de datos empresariales es menos detallado que las bases de datos individuales que comprende. Este modelo se concentra en los temas principales de las bases de datos operativas, en lugar de hacerlo en los detalles. Un modelo de datos empresariales se puede desarrollar para la planeación de los datos (qué bases de datos desarrollar) o el apoyo a las decisiones (cómo integrar y resumir las bases de datos existentes). La sección 14.3 describe los detalles de la planeación de datos, mientras que el capítulo 16 describe los detalles del desarrollo de un modelo de datos empresariales para el apoyo en la toma de decisiones. Por lo general, el administrador de datos participa en ambos esfuerzos.

Las grandes organizaciones pueden ofrecer una gran cantidad de especialización en la administración de datos y de bases de datos. Para la administración de datos, la especialización puede ser por tarea y entorno. En cuanto a las tareas, los administradores de datos se pueden especializar en la planeación en lugar del establecimiento de políticas. En cuanto al entorno, se especializan en entornos como el apoyo a las decisiones, el procesamiento de transacciones y los datos no tradicionales, como imágenes, texto y video. Para la administración de bases de datos, la especialización puede ocurrir por DBMS, tarea y entorno. Debido a la dificultad para aprender un DBMS, por lo regular los DBA se especializan en un producto o en unos cuantos. La especialización por tarea casi siempre se divide entre el procesamiento de transacciones y los *data warehouses*.

En las pequeñas organizaciones, la frontera entre la administración de datos y de bases de datos no es muy estricta. Es probable que no existan puestos separados para administradores de datos y de bases de datos, sino que la misma persona cubra las funciones de ambos. Conforme las organizaciones aumentan de tamaño, la especialización casi siempre evoluciona hasta que se crean puestos separados.

## 14.2 Herramientas de administración de bases de datos

Para cumplir con las responsabilidades mencionadas en la sección anterior, los administradores de bases de datos utilizan diversas herramientas. Ya aprendió qué herramientas se usan para el modelado de datos, diseño de bases de datos lógicos, creación de vistas, diseño físico de bases de datos, disparadores y procedimientos almacenados. Algunas de las herramientas son sentencias SQL (CREATE VIEW y CREATE INDEX), mientras otras son parte de las herramientas CASE para el desarrollo de bases de datos. Esta sección presenta herramientas adicionales para seguridad, integridad y acceso a diccionarios de datos, además de analizar el manejo de procedimientos almacenados y disparadores.

### 14.2.1 Seguridad

#### seguridad de bases de datos

proteger las bases de datos del acceso no autorizado y la destrucción maliciosa.

#### reglas de autorización

definir usuarios autorizados, operaciones permitidas y partes accesibles de una base de datos. El sistema de seguridad de bases de datos almacena las reglas de autorización y las pone en práctica para el acceso a cada base de datos.

#### control de acceso discrecional

los usuarios tienen derechos de acceso asignados o privilegios para partes específicas de una base de datos. El control de acceso discrecional es el tipo de control de seguridad más común para el que los DBMS comerciales ofrecen soporte.

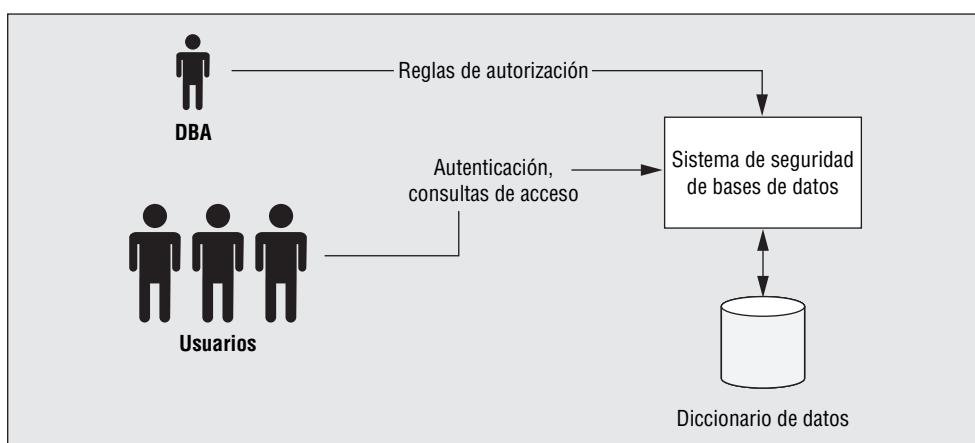
La seguridad comprende la protección de una base de datos del acceso no autorizado y de la destrucción maliciosa. Debido al valor de la información en las bases de datos corporativas, existe una gran motivación para los usuarios no autorizados que tratan de obtener acceso ilegal a ellas. Los competidores se sienten atraídos por el acceso a información delicada sobre los planes de desarrollo de los productos, las iniciativas para ahorrar en costos y los perfiles de los clientes. Los criminales pretenden hurtar resultados financieros no anunciados, transacciones empresariales e información delicada sobre los clientes, como números de tarjetas de crédito. Los inadaptados sociales y terroristas pueden causar estragos con la destrucción intencional de los registros en las bases de datos. Con el uso cada vez más frecuente del World Wide Web para realizar negocios, competidores, criminales e inadaptados sociales tienen más oportunidades de poner en riesgo la seguridad de las bases de datos.

La seguridad es un amplio tema que comprende muchas disciplinas. Existen aspectos legales y éticos relacionados con quién tiene acceso a los datos y cuándo se pueden revelar. Hay controles de red, de hardware, de sistema operativo y físicos que aumentan los controles que ofrecen los DBMS. Además, existen problemas operativos relacionados con contraseñas, dispositivos de autenticación y privacidad. No nos ocupamos de estos problemas porque se encuentran más allá del alcance de los DBMS y especialistas en bases de datos. El resto de esta subsección enfatiza los enfoques para controlar el acceso y las sentencias SQL para las reglas de autorización.

Para controlar el acceso, los DBMS ofrecen soporte para la creación y almacenamiento de reglas de autorización, así como para su aplicación cuando los usuarios acceden a una base de datos. La figura 14.4 ilustra la interacción de estos elementos. Los administradores de bases de datos crean reglas de autorización que definen quién tiene acceso a qué partes de una base de datos para realizar qué operaciones. La aplicación de las reglas de autorización comprende la autenticación del usuario y la garantía de que las consultas de acceso no van a violar las reglas de autorización (recuperaciones y modificaciones a las bases de datos). La autenticación ocurre cuando un usuario se conecta por primera vez a un DBMS. Es preciso revisar las reglas de autorización para cada solicitud de acceso.

El planteamiento más común para las reglas de autorización se conoce como control de acceso discrecional. En éste, a los usuarios se les asignan derechos o privilegios de acceso a secciones específicas de una base de datos. Para un control preciso, los privilegios se especifican para las vistas en lugar de las tablas o los campos. Es posible dar a los usuarios la habilidad de leer, actualizar, insertar y eliminar partes específicas de una base de datos. Para simplificar el mantenimiento de las reglas de autorización, es posible asignar privilegios a los grupos o funciones, en lugar de a los usuarios individuales. Como las funciones son más estables que los usuarios individuales, las reglas de autorización que hacen referencia a las funciones requieren menos mantenimiento que las reglas que hacen referencia a usuarios individuales. A los usuarios se les asignan funciones y una contraseña. Durante el proceso de registro a la base de datos, el sistema de seguridad autentica a los usuarios e indica a qué funciones pertenecen.

**FIGURA 14.4**  
Sistema de seguridad de bases de datos



**control de acceso obligatorio**

un planteamiento de seguridad para bases de datos muy delicadas y estáticas. Un usuario tiene acceso a un elemento de la base de datos si el nivel de autorización del usuario proporciona acceso al nivel de clasificación del elemento.

Los controles de acceso obligatorios son menos flexibles que los controles de acceso discretionarios. En los primeros, a cada objeto se le asigna un nivel de clasificación y a cada usuario se le da un nivel de autorización. Un usuario puede tener acceso a un objeto si su nivel de autorización ofrece acceso al nivel de clasificación del objeto. Los niveles de autorización y clasificación típicos son confidencial, secreto y súper secreto. Los planteamientos de control de acceso obligatorio se aplican sobre todo a bases de datos muy delicadas y estáticas relacionadas con la defensa nacional y la recopilación de inteligencia. En vista de la flexibilidad limitada de los controles obligatorios de acceso, sólo unos cuantos DBMS ofrecen soporte para ellos. Sin embargo, los DBMS que se utilizan en la defensa nacional y la recopilación de información deben ofrecer soporte para los controles obligatorios.

Además de estos controles de acceso, los DBMS ofrecen soporte para la encriptación de bases de datos. La encriptación comprende la codificación de los datos para ocultar su significado. Un algoritmo de encriptación cambia los datos originales (conocidos como texto plano o plaintext). Para descifrarlos, el usuario proporciona una llave de codificación que le permite restaurar los datos codificados (conocidos como ciphertext) al formato original (texto plano). Dos de los algoritmos de encriptación más populares son el estándar de encriptación de datos (*Data Encryption Standard*) y el algoritmo de encriptación con llaves públicas (Public-Key Encryption). Como el estándar de encriptación de datos se puede romper utilizando el poder de la computación masiva, el algoritmo de encriptación con llaves públicas se ha convertido en el preferido.

*Sentencias de seguridad de SQL:2003*

SQL:2003 ofrece soporte para reglas de autorización discretionarias que utilizan sentencias CREATE/DROP ROLE y sentencias GRANT/REVOKE. Al crear una función, el DBMS otorga la función al usuario actual o a la función actual. En el ejemplo 14.1, las funciones ISFaculty e ISAdvisor se otorgan al usuario actual, mientras que la función ISAdministrator se otorga a la función del usuario actual. La cláusula WITH ADMIN significa que el usuario al que se le asigna la función puede asignarla a otros. La opción WITH ADMIN se debe utilizar con precaución porque ofrece una gran libertad para el rol. Es posible retirar una función con la sentencia DROP ROLE.

**EJEMPLO 14.1  
(SQL:2003)****Ejemplos de la sentencia CREATE ROLE**

```
CREATE ROLE ISFaculty;
CREATE ROLE ISAdministrator WITH ADMIN CURRENT_ROLE;
CREATE ROLE ISAdvisor;
```

En una sentencia GRANT, usted especifica los privilegios (ver tabla 14.3), el objeto (tabla, columna o vista) y la lista de usuarios autorizados (o funciones). En el ejemplo 14.2, se ofrece acceso SELECT a tres funciones (ISFaculty, ISAdvisor, ISAdministrator), mientras que el acceso UPDATE sólo se da a ISAdministrator. A los usuarios individuales se les deben asignar las funciones antes de que tengan acceso a la vista *ISSStudentGPA*.

**TABLA 14.3**  
Explicación de los privilegios comunes en SQL:2003

Privilegio	Explicación
SELECT	Consulta el objeto; puede especificarse para columnas individuales.
UPDATE	Modifica el valor; puede especificarse para columnas individuales.
INSERT	Agrega una nueva fila; puede especificarse para columnas individuales.
DELETE	Elimina una fila; puede especificarse para columnas individuales.
TRIGGER	Crea un disparador para una tabla específica.
REFERENCES	Hace referencia a las columnas de una tabla determinada con limitaciones de integridad.
EXECUTE	Ejecuta el procedimiento almacenado.

**EJEMPLO 14.2  
(SQL:2003)**

```

Definición de vista, sentencias GRANT y REVOKE

CREATE VIEW ISSStudentGPA AS
    SELECT StdSSN, StdFirstName, StdLastName, StdGPA
        FROM Student
       WHERE StdMajor = 'IS';
-- Grant privileges to roles
GRANT SELECT ON ISSStudentGPA
    TO ISFaculty, ISAdvisor, ISAdministrator;
GRANT UPDATE ON ISSStudentGPA.StdGPA TO ISAdministrator;
-- Assign users to roles
GRANT ISFaculty TO Mannino;
GRANT ISAdvisor TO Olson;
GRANT ISAdministrator TO Smith WITH GRANT OPTION;

REVOKE SELECT ON ISSStudentGPA FROM ISFaculty RESTRICT;

```

El sentencia GRANT también se puede usar para asignar usuarios a los roles, como se muestra en las últimos tres sentencias GRANT del ejemplo 14.2. Además de otorgar los privilegios en la tabla 14.3, es posible autorizar a un usuario que pase los privilegios a otros usuarios empleando la palabra clave WITH GRANT OPTION. En la última sentencia GRANT del ejemplo 14.2, el usuario Smith puede otorgar la función ISAdministrator a otros usuarios. La opción WITH GRANT se debe utilizar con precaución porque ofrece mucha libertad al usuario.

La sentencia REVOKE se utiliza para eliminar un privilegio de acceso. En la última sentencia del ejemplo 14.2, se elimina el privilegio SELECT de ISFaculty. La cláusula RESTRICT significa que el privilegio se revoca sólo si no se otorgó a la función especificada por más de un usuario.

*Seguridad en Oracle y Access*

Oracle 10g amplía las sentencias de seguridad de SQL:2003 con la sentencia CREATE USER (crear usuario), funciones predefinidas y privilegios adicionales. En SQL:2003, la creación de usuarios es un problema de implementación. Como Oracle no depende del sistema operativo para la creación de usuarios, proporciona la sentencia CREATE USER. Oracle ofrece funciones predefinidas para usuarios con altos privilegios que incluyen la función CONNECT (conectar) para crear tablas en un esquema, la función RESOURCE para crear tablas y objetos de aplicaciones como procedimientos almacenados, y la función DBA para administrar bases de datos. Para los privilegios, Oracle distingue entre los privilegios del sistema (independientes del objeto) y los privilegios del objeto. Por lo regular, el hecho de otorgar privilegios de objetos está reservado para funciones altamente seguras debido al largo alcance de los privilegios del sistema, como muestra la tabla 14.4. Los privilegios de objeto de Oracle son similares a los de SQL:2003, sólo que Oracle proporciona más objetos que SQL:2003, como muestra la tabla 14.5.

La mayoría de los DBMS permiten restricciones de autorización por objetos de aplicaciones, como formularios y reportes, además de los objetos de bases de datos permitidos en la sentencia GRANT. Estas limitaciones adicionales de seguridad casi siempre se especifican en las interfaces propietarias o en las herramientas de desarrollo de aplicaciones, en lugar de SQL. Por ejemplo, Microsoft Access 2003 permite la definición de reglas de autorización a través de la ventana permisos para usuarios y grupos, como se ilustra en la figura 14.5. Utilizando esta ventana se pueden especificar los permisos para los objetos de bases de datos (tablas y consultas almacenadas), así como los objetos de aplicaciones (formularios y reportes). Además, Access SQL ofrece soporte para las sentencias GRANT/REVOKE de manera similar a las sentencias de SQL:2003, así como para las sentencias CREATE/DROP para usuarios y grupos.

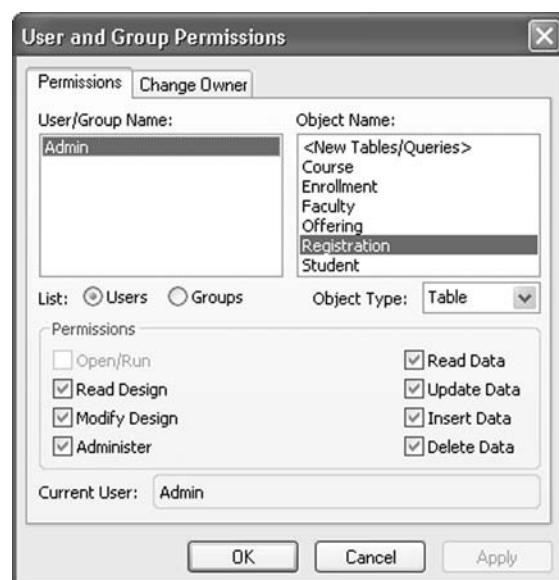
**TABLA 14.4**  
Explicación de los privilegios de sistema comunes en Oracle

Privilegio de sistema	Explicación
CREATE X, CREATE ANY X	Crea objetos tipo X en el propio esquema; CREATE ANY permite la creación de objetos en otros esquemas <sup>1</sup>
ALTER X, ALTER ANY X	Altera objetos tipo X en el propio esquema; ALTER ANY X permite alterar objetos en otros esquemas
INSERT ANY, DELETE ANY,	Inserta, elimina, actualiza y selecciona de una tabla en cualquier esquema
UPDATE ANY, SELECT ANY	
DROP X, DROP ANY X	RETIRA objetos tipo X en el propio esquema; DROP ANY permite retirar objetos en otros esquemas
ALTER SYSTEM, ALTER DATABASE, ALTER SESSION	Emite comandos ALTER SYSTEM, comandos ALTER DATABASE y comandos ALTER SESSION
ANALYZE ANY	Analiza cualquier tabla, índice o grupo

**TABLA 14.5**  
Diagramación entre los privilegios y objetos comunes de Oracle

Privilegio/ Objeto	Tabla	Vista	Secuencia <sup>2</sup>	Procedimiento, función, paquete, biblioteca, operador, tipo de índice	Vista materializada <sup>3</sup>
ALTER	X		X		
DELETE	X	X			X
EXECUTE				X	
INDEX	X				
INSERT	X	X			X
REFERENCES	X	X			
SELECT	X	X	X		X
UPDATE	X	X			X

**FIGURA 14.5**  
Ventana de permisos para usuarios y grupos en Microsoft Access 2003



<sup>1</sup> Un esquema es un grupo de tablas relacionadas y otros objetos de Oracle que se manejan como una unidad.

<sup>2</sup> Una secuencia es un grupo de valores que mantiene Oracle. Por lo regular, las secuencias se utilizan para llaves primarias generadas por el sistema.

<sup>3</sup> Una vista materializada se guarda en lugar de derivarse. Las vistas materializadas son útiles en los *data warehouses*, como se presenta en el capítulo 16.

## 14.2.2 Restricciones de integridad

Usted ya conoce las restricciones de integridad que presentamos en capítulos anteriores. En el capítulo 3, le presentamos las llaves primarias, las llaves foráneas, las llaves candidatas y las restricciones de no nulidad, además de la sintaxis SQL correspondiente. En el capítulo 5, estudió las restricciones de cardinalidad y las restricciones de la jerarquía de generalización. En el capítulo 7, estudió las dependencias funcionales y de valores múltiples como parte del proceso de normalización. Además, el capítulo 8 describió índices que se pueden usar para aplicar con eficiencia las llaves primarias y candidatas. Esta subsección describe otras clases de restricciones de integridad y la sintaxis SQL correspondiente.

### *Dominios SQL*

En el capítulo 3 definimos los tipos de datos SQL estándar. Un tipo de datos indica la clase de datos (caracteres, numéricos, sí/no, etc.) y las operaciones permitidas (operaciones numéricas, operaciones de hileras, etc.) para las columnas que utilizan el tipo de datos. SQL:2003 ofrece la capacidad limitada de definir nuevos tipos de datos utilizando la sentencia CREATE DOMAIN. Un dominio se puede crear como un subconjunto de un tipo de datos estándar. El ejemplo 14.3 demuestra la sentencia CREATE DOMAIN, además del uso de los dominios nuevos en lugar de los tipos de datos estándar. La cláusula CHECK define una limitación para el dominio limitándolo a un subconjunto de tipos de datos estándar.

**EJEMPLO 14.3  
(SQL:2003)**
**Sentencias CREATE DOMAIN y uso de los dominios**

```
CREATE DOMAIN StudentClass AS CHAR(2)
    CHECK ( VALUE IN ('FR', 'SO', 'JR', 'SR') )

CREATE DOMAIN CourseUnits AS SMALLINT
    CHECK ( VALUE BETWEEN 1 AND 9 )
```

En CREATE TABLE para la tabla *Student*, es posible hacer referencia al dominio en la columna *StdClass*.

StdClass StudentClass NOT NULL

En CREATE TABLE para la tabla *Course*, es posible hacer referencia al dominio en la columna *CrsUnits*.

CrsUnits CourseUnits NOT NULL

SQL:2003 ofrece una característica relacionada conocida como tipo distintivo. Como en el caso de un dominio, un tipo distintivo no puede tener limitaciones. Sin embargo, la especificación SQL ofrece una mejor verificación de los tipos distintivos en comparación con los dominios. Una columna que tiene un tipo distintivo se puede comparar sólo con otra columna que utiliza el mismo tipo. El ejemplo 14.4 muestra las definiciones de los tipos distintivos y una comparación entre las columnas con base en los tipos.

**EJEMPLO 14.4  
(SQL:2003)**
**Tipos distintivos y uso de los tipos distintivos**

```
-- USD distinct type and usage in a table definition
CREATE DISTINCT TYPE USD AS DECIMAL(10,2);
USProdPrice USD

CREATE DISTINCT TYPE Euro AS DECIMAL(10,2);
EuroProdPrice Euro

-- Type error: columns have different distinct types
USProdPrice > EuroProdPrice
```

Para las bases de datos orientadas a objetos, SQL:2003 ofrece tipos definidos por el usuario, una capacidad más poderosa que los dominios o tipos distintivos. Los tipos de datos definidos por el usuario se pueden definir con operadores y funciones nuevas. Además, los tipos de datos definidos por el usuario se pueden definir utilizando otros tipos de datos definidos por el usuario. El capítulo 18 describe los tipos de datos definidos por el usuario como parte de la presentación de las características de SQL:2003 orientadas a los objetos. Gracias a las limitaciones, la mayor parte de los DBMS ya no ofrecen soporte para dominios y tipos distintivos. Por ejemplo, Oracle 10g soporta tipos definidos por el usuario, pero no dominios ni tipos distintivos.

#### *Restricciones CHECK en la sentencia CREATE TABLE*

La restricción CHECK puede usarse cuando una restricción comprende las condiciones de fila en las columnas de la misma tabla. Estas restricciones se especifican como parte de la sentencia CREATE TABLE, como muestra el ejemplo 14.5. Para un registro más sencillo, siempre debe dar nombre a las restricciones. Al ocurrir una violación a las restricciones, casi todos los DBMS despliegan el nombre de la restricción.

#### **EJEMPLO 14.5 (SQL:2003)**

##### **Cláusulas de la restricción CHECK**

Ésta es una sentencia CREATE TABLE con restricciones CHECK para el rango GPA válido y estudiantes de clases superiores (segundo y tercer año) que tienen una materia principal declarada (no nula).

##### **CREATE TABLE Student**

```
( StdSSN      CHAR(11),
  StdFirstName VARCHAR(50) CONSTRAINT StdFirstNameRequired NOT NULL,
  StdLastName   VARCHAR(50) CONSTRAINT StdLastNameRequired NOT NULL,
  StdCity       VARCHAR(50) CONSTRAINT StdCityRequired NOT NULL,
  StdState      CHAR(2)      CONSTRAINT StdStateRequired NOT NULL,
  StdZip        CHAR(9)      CONSTRAINT StdZipRequired NOT NULL,
  StdMajor      CHAR(6),
  StdClass      CHAR(6),
  StdGPA        DECIMAL(3,2),
  CONSTRAINT PKStudent PRIMARY KEY (StdSSN),
  CONSTRAINT ValidGPA CHECK ( StdGPA BETWEEN 0 AND 4 ),
  CONSTRAINT MajorDeclared CHECK
    ( StdClass IN ('FR','SO') OR StdMajor IS NOT NULL ) )
```

Aunque las restricciones CHECK son ampliamente soportadas, casi todos los DBMS limitan las condiciones dentro de ellas. La especificación SQL:2003 permite cualquier condición que pueda aparecer en una sentencia SELECT, incluidas aquellas que comprenden sentencias SELECT. Casi ningún DBMS permite condiciones que comprendan sentencias SELECT en una restricción CHECK. Por ejemplo, Oracle 10g prohíbe las sentencias SELECT en las restricciones CHECK, así como las referencias a las columnas de otras tablas. Para estas limitaciones complejas es posible utilizar afirmaciones (si el DBMS las soporta) o disparadores, en caso de que las afirmaciones no cuenten con soporte.

#### *Afirmaciones (Assertions) SQL:2003*

Las afirmaciones SQL:2003 son más poderosas que las restricciones relacionadas con los dominios, columnas, llaves primarias y llaves foráneas. A diferencia de las restricciones CHECK, las afirmaciones no están relacionadas con una tabla específica. Una afirmación puede comprender una sentencia SELECT de complejidad arbitraria. Por tanto, las afirmaciones se pueden usar para las restricciones que comprenden varias tablas y cálculos estadísticos, como muestran los

ejemplos 14.6 a 14.8. Sin embargo, las afirmaciones complejas se deben utilizar con cuidado porque su uso puede ser deficiente. Puede haber formas más eficientes de emplear las afirmaciones, como a través de condiciones de eventos en un formulario y procedimientos almacenados. Como DBA, le sugerimos investigar las capacidades para la programación de eventos de las herramientas de desarrollo de aplicaciones, antes de utilizar afirmaciones complejas.

Las afirmaciones se verifican después de terminar las operaciones de modificación relacionadas. Por ejemplo, revisaríamos la afirmación *OfferingConflict* en el ejemplo 14.7 para verificar cada inserción de una fila *Offering* y cada cambio a una de las columnas en la cláusula

#### EJEMPLO 14.6 (SQL:2003)

##### Sentencia CREATE ASSERTION

Esta sentencia de afirmación garantiza que cada profesor tiene una carga de cursos entre tres y nueve unidades.

```
CREATE ASSERTION FacultyWorkLoad
  CHECK (NOT EXISTS
    ( SELECT Faculty.FacSSN, OffTerm, OffYear
      FROM Faculty, Offering, Course
      WHERE Faculty.FacSSN = Offering.FacSSN
        AND Offering.CourseNo = Course.CourseNo
      GROUP BY Faculty.FacSSN, OffTerm, OffYear
      HAVING SUM(CrsUnits) < 3 OR SUM(CrsUnits) > 9 ) )
```

#### EJEMPLO 14.7 (SQL:2003)

##### Sentencia CREATE ASSERTION

Esta sentencia de afirmación garantiza que no hay dos cursos que se den a la misma hora y en el mismo lugar. Es preciso refinar las condiciones que comprenden las columnas *OffTime* y *OffDays* para verificar que no se superpongan, y no sólo para revisar que sean iguales. Estas mejoras no se muestran porque comprenden las funciones hilera y fecha específicas de un DBMS.

```
CREATE ASSERTION OfferingConflict
  CHECK (NOT EXISTS
    ( SELECT O1.OfferNo
      FROM Offering O1, Offering O2
      WHERE O1.OfferNo <> O2.OfferNo
        AND O1.OffTerm = O2.OffTerm
        AND O1.OffYear = O2.OffYear
        AND O1.OffDays = O2.OffDays
        AND O1.OffTime = O2.OffTime
        AND O1.OffLocation = O2.OffLocation ) )
```

#### EJEMPLO 14.8 (SQL:2003)

##### Sentencia de afirmación para garantizar que los estudiantes de tiempo completo tengan por lo menos nueve unidades

```
CREATE ASSERTION FullTimeEnrollment
  CHECK (NOT EXISTS
    ( SELECT Enrollment.RegNo
      FROM Registration, Offering, Enrollment, Course
      WHERE Offering.OfferNo = Enrollment.OfferNo
        AND Offering.CourseNo = Course.CourseNo
        AND Offering.RegNo = Registration.RegNo
        AND RegStatus = 'F'
      GROUP BY Enrollment.RegNo
      HAVING SUM(CrsUnits) >= 9 ) )
```

WHERE de la afirmación. En algunos casos, es preciso demorar una afirmación hasta terminar otras sentencias. Podemos utilizar la palabra clave DEFERRABLE para permitir la prueba de una afirmación al final de una transacción, en lugar de hacerlo de manera inmediata. La revisión diferida es un aspecto del diseño de transacciones que estudiaremos en el capítulo 15.

Las afirmaciones no tienen soporte con mucha frecuencia porque se superponen con los disparadores. Una afirmación es un tipo de disparador limitado con una condición y una acción implícita. Como las afirmaciones son más sencillas que los disparadores, casi siempre son más fáciles de crear y más eficientes de ejecutar. Sin embargo, ningún gran DBMS relacional ofrece soporte para las afirmaciones, de modo que es necesario usar disparadores en los casos en los que serían apropiadas las afirmaciones.

### 14.2.3 Administración de disparadores y procedimientos almacenados

En el capítulo 11, aprendió los conceptos y detalles de encriptación de los procedimientos almacenados y disparadores. Aunque un DBA escribe procedimientos almacenados y disparadores para ayudar en la administración de las bases de datos, las principales responsabilidades de un DBA son manejar esos procedimientos y disparadores, y no escribirlos. Sus responsabilidades también incluyen establecer los estándares para las prácticas de encriptación, vigilar las dependencias y entender las interacciones de los disparadores.

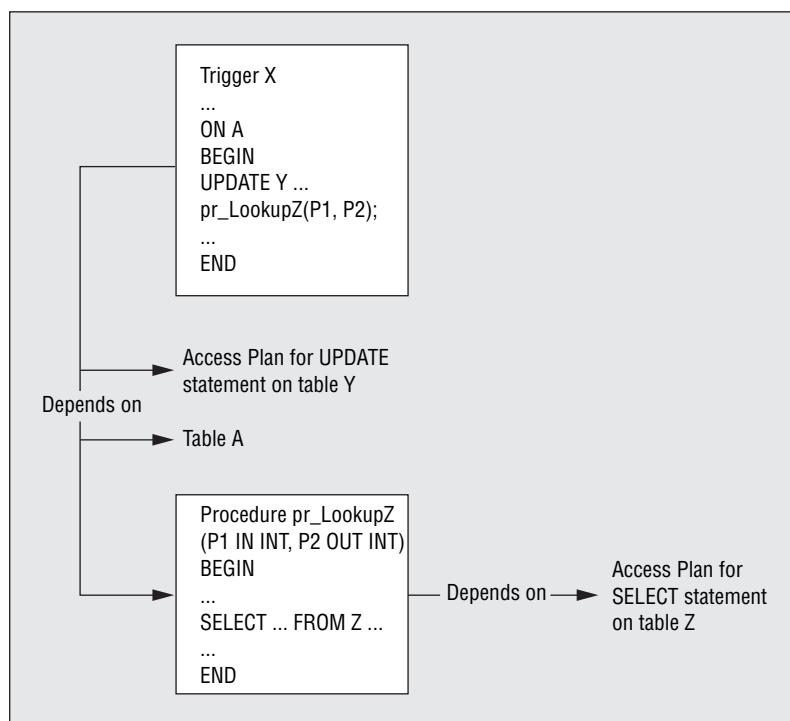
Para las prácticas de encriptación, un DBA debe considerar los estándares de documentación, el uso de parámetros y el contenido, como resume la tabla 14.6. Los estándares de documentación pueden incluir estándares de nombres, explicaciones de parámetros y descripciones de las condiciones previas y posteriores a los procedimientos. Es preciso vigilar el uso de parámetros en los procedimientos y funciones. Las funciones sólo deben utilizar parámetros de entrada y no deben tener efectos secundarios. Para el contenido, los disparadores no deben realizar revisiones de integridad que se puedan codificar como restricciones de integridad declarativa (restricciones CHECK, llaves primarias, llaves foráneas...). Para reducir el mantenimiento, los disparadores y los procedimientos almacenados deben hacer referencia a los tipos de datos de las columnas de las bases de datos asociadas. En Oracle, esta práctica comprende tipos de datos anclados. Como la mayor parte de las herramientas de desarrollo de aplicaciones ofrecen soporte para disparadores y procedimientos de eventos para formularios y reportes, la elección entre un disparador/procedimiento de base de datos y un disparador/procedimiento de aplicación no siempre está clara. Un DBA debe participar en el establecimiento de los estándares que ofrecen una guía entre el uso de disparadores y procedimientos de bases de datos y el uso de disparadores y procedimientos de aplicaciones.

Un procedimiento almacenado o un disparador depende de las tablas, vistas, procedimientos y funciones a los que hace referencia, así como de los planes de acceso que crea el compilador SQL. Cuando cambia un objeto al que se hace referencia, es necesario volver a compilar los elementos que dependen de él. En la figura 14.6, el disparador X necesita una recopilación si se llevan a cabo cambios al plan de acceso para la sentencia UPDATE en el cuerpo del disparador. De modo similar, el procedimiento necesita una recopilación si se realizan cambios a la tabla A o al procedimiento pr\_LookupZ. La mayor parte de los DBMS mantienen dependencias para

**TABLA 14.6**  
Resumen de preocupaciones para un DBA en la práctica de encriptación

Área de la práctica de encriptación	Preocupaciones
Documentación	Estándares de nombres para procedimientos y disparadores; comentarios que describen las condiciones anteriores y posteriores
Uso de parámetros	Sólo parámetros de entrada para las funciones; sin efectos secundarios para las funciones
Contenido de los disparadores y los procedimientos	No utilice disparadores para las limitaciones de integridad estándar; uso de tipos de datos anclados para las variables; estándares para los disparadores de aplicaciones y los procedimientos de eventos en comparación con los disparadores y procedimientos de bases de datos

**FIGURA 14.6**  
Dependencias entre  
objetos de una base  
de datos



**TABLA 14.7**  
Resumen de preocu-  
paciones de depen-  
dencia para un DBA

Área de dependencia	Preocupaciones
Obsolescencia del plan de acceso	Los DBMS deben recompilar automáticamente. Es probable que el DBA necesite recompilar cuando las estadísticas del optimizado ya no estén actualizadas.
Modificación de objetos referenciados	Los DBMS deben recompilar automáticamente. El DBA debe elegir entre el mantenimiento del sello de la hora y la firma para los procedimientos y funciones remotos.
Eliminación de objetos referenciados	El DBMS marca el procedimiento/disparador como inválido si se eliminan los objetos referenciados.

garantizar que los procedimientos almacenados y disparadores funcionan de manera correcta. Si un procedimiento o disparador utiliza una sentencia SQL, casi todos los DBMS recopilan automáticamente este procedimiento o disparador, en caso de que el plan de acceso asociado sea obsoleto.

Un DBA debe estar consciente de las limitaciones de las herramientas que ofrece el DBMS para el manejo de dependencias. La tabla 14.7 resume los aspectos de la administración de dependencias de la obsolescencia del plan de acceso, la modificación de los objetos referenciados y la eliminación de los objetos referenciados. Para los planes de acceso, un DBA debe entender que quizás sea necesaria la recompilación manual, si las estadísticas del optimizador dejan de estar actualizadas. Para procedimientos y funciones almacenados en un sitio remoto, un DBA puede elegir entre el mantenimiento de la dependencia con sello de la hora o la firma. Con el mantenimiento del sello de la hora, un DBMS recompila un objeto dependiente para cualquier cambio en los objetos referenciados. El mantenimiento del sello de la hora puede dar lugar a una recompilación excesiva, porque muchos de los cambios a los objetos referenciados no requieren de la recompilación de los objetos dependientes. El mantenimiento de firma comprende la recompilación cuando cambia una firma (nombre de parámetro o uso). Asimismo, un DBA debe estar consciente de que un DBMS no va a recompilar un procedimiento ni un disparador si se elimina uno de los objetos referenciados. El procedimiento o disparador dependiente se marcará como inválido porque la recompilación no es posible.

Las interacciones de los disparadores se estudian en el capítulo 11, como parte de los procedimientos de ejecución de los disparadores. Estos últimos interactúan cuando un disparador activa otros disparadores, y cuando éstos se superponen llevando a la activación en orden arbitrario. Un DBA puede usar las herramientas de análisis de disparadores que proporciona un fabricante de DBMS o, en caso de no contar con ninguna herramienta, analizar manualmente las interacciones de los disparadores. Un DBA deberá realizar pruebas adicionales para los disparadores que interactúan entre sí. Para minimizar la interacción de los disparadores, un DBA debe implementar lineamientos como los que resume la tabla 14.8.

#### 14.2.4 Manipulación del diccionario de datos

##### **metadatos**

datos que describen otros datos, incluyendo su fuente, uso, valor y significado.

El diccionario de datos es una base de datos especial que describe bases de datos individuales y su entorno. El diccionario de datos contiene descriptores de datos llamados **metadatos**, que definen la fuente, el uso, el valor y el significado de los datos. Por lo general, los DBA manejan dos tipos de diccionarios de datos para registrar el entorno de la base de datos. Cada DBMS ofrece un diccionario de datos para registrar tablas, columnas, afirmaciones, índices y otros objetos que maneja. Las herramientas CASE independientes ofrecen un diccionario de datos conocido como diccionario de recursos de información, que registra un amplio rango de objetos relacionados con el desarrollo de sistemas de información. Esta subsección ofrece detalles acerca de ambos tipos de diccionarios de datos.

##### *Tablas de catálogo en SQL:2003 y Oracle*

SQL:2003 contiene las tablas del catálogo en el Definition\_Schema, como resume la tabla 14.9. El Definition\_Schema contiene una o más tablas del catálogo que corresponden a cada objeto que se puede crear en una definición de datos SQL o una sentencia de control de datos. Las tablas de catálogo de la base en el Definition\_Schema no están pensadas para accederse en las aplicaciones. Para tener acceso a los metadatos en las aplicaciones, SQL:2003 ofrece el Information\_Schema que contiene vistas de las tablas de catálogo de la base del Definition\_Schema.

Definition\_Schema e Information\_Schema de SQL:2003 tienen pocas implementaciones porque la mayor parte de los DBMS ya tenían tablas de catálogo patentadas mucho antes del lanzamiento del estándar. Por tanto, será necesario que aprenda las tablas de catálogo de todos los DBMS con los que usted trabaja. Por lo regular, un DBMS puede tener cientos de tablas de

**TABLA 14.8**  
Resumen de lineamientos para controlar la complejidad de los disparadores

Lineamiento	Explicación
Disparadores BEFORE ROW	No utiliza sentencias de manipulación de datos en los disparadores BEFORE ROW para evitar la activación de otros disparadores.
Disparadores UPDATE	Utiliza una lista de columnas para los disparadores UPDATE con el fin de reducir la superposición de los disparadores.
Acciones en las filas referenciadas	Tenga cuidado con los disparadores en las tablas que se ven afectadas por las acciones en las filas referenciadas. Estos disparadores se activarán como resultado de las acciones en las tablas madre.
Disparadores superpuestos	No dependen de un orden de activación específico.

**TABLA 14.9**  
Resumen de las tablas de catálogo más importantes en SQL:2003

Tabla	Contenido
USERS	Una fila para cada usuario.
DOMAINS	Una fila para cada dominio.
DOMAIN_CONSTRAINTS	Una fila para cada limitación de dominio en una tabla.
TABLES	Una fila para cada tabla y vista.
VIEWS	Una fila para cada vista.
COLUMNS	Una fila para cada columna.
TABLE_CONSTRAINTS	Una fila para cada limitación en la tabla.
REFERENTIAL_CONSTRAINTS	Una fila para cada limitación de referencia.

**TABLA 14.10**

**Tablas comunes de catálogo para Oracle**

Nombre de tabla	Contenido
USER_CATALOG	Contiene datos básicos sobre cada tabla y vista definidas por un usuario.
USER_OBJECTS	Contiene datos sobre cada objeto (funciones, procedimientos, índices, disparadores, afirmaciones, etc.) definido por un usuario. Esta tabla contiene la hora de creación y la última vez que cambió para cada objeto.
USER_TABLES	Contiene datos ampliados acerca de cada tabla, como la distribución del espacio y los resúmenes estadísticos.
USER_TAB_COLUMNS	Contiene datos básicos y ampliados para cada columna, como nombre de columna, referencia de la tabla, tipo de datos y un resumen estadístico.
USER_VIEWS	Contiene la sentencia SQL que define cada vista.

catálogo. Sin embargo, para cualquier tarea específica, como manejar disparadores, un DBA necesita utilizar una pequeña cantidad de tablas de catálogo. La tabla 14.10 presenta algunas de las tablas de catálogo más importantes de Oracle.

Un DBA modifica en forma implícita las tablas de catálogo cuando utiliza comandos de definición de datos, como la sentencia CREATE TABLE. Los DBMS usan las tablas de catálogo para procesar consultas, autorizar usuarios, revisar las limitaciones de integridad y llevar a cabo el procesamiento de otras bases de datos. El DBMS consulta las tablas de catálogo antes de emprender cada acción. Por tanto, la integridad de estas tablas es crucial para la operación del DBMS. Sólo los usuarios más autorizados deben tener permitido modificar las tablas de catálogo. Para aumentar la seguridad y la confiabilidad, el diccionario de datos casi siempre es una base de datos separada y guardada de manera independiente de las bases de datos de los usuarios.

Un DBA puede consultar las tablas de catálogo a través de interfaces propietarias y sentencias SELECT. Las interfaces propietarias, como la ventana Table Definition de Microsoft Access y el Oracle Enterprise Manager, son más fáciles de usar que SQL, pero no son compatibles entre los DBMS. Las sentencias SELECT ofrecen mayor control sobre la información recuperada que las interfaces propietarias.

### *Diccionario de recursos de información*

#### **diccionario de recursos de información**

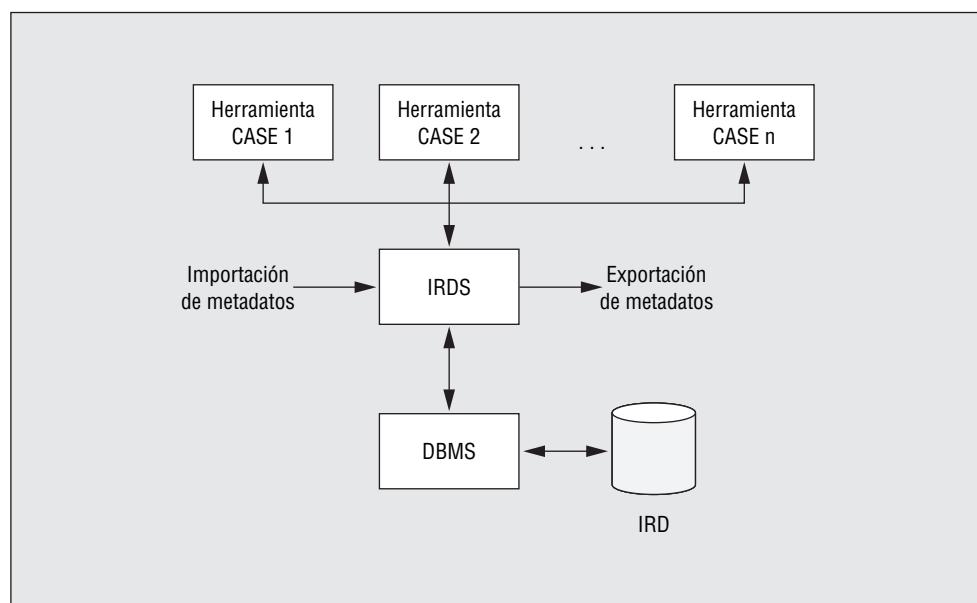
una base de datos de metadatos que describe todo el ciclo de vida de los sistemas de información. El sistema de diccionarios de recursos de información administra el acceso a un IRD.

Un diccionario de recursos de información contiene una colección mucho más extensa de metadatos que un diccionario de datos para un DBMS. Un diccionario de recursos de información (IRD) contiene metadatos acerca de las bases de datos individuales, los procesos humanos y computarizados, el manejo de configuraciones, el control de las versiones, los recursos humanos y el entorno de cómputo. Conceptualmente, un IRD define los metadatos utilizados durante todo el ciclo de vida de los sistemas de información. Tanto los DBA como los DA pueden utilizar un IRD para administrar los recursos de información. Además, otros profesionales de los sistemas de información pueden usar un IRD durante tareas seleccionadas en el ciclo de vida de los sistemas de información.

Debido a su amplia función, un DBMS no consulta un IRD para realizar operaciones. En vez de ello, un sistema de diccionarios de recursos de información (IRDS) administra el IRD. Muchas herramientas CASE pueden utilizar el IRDS para tener acceso a un IRD, como ilustra la figura 14.7. Las herramientas CASE tienen acceso a un IRD directamente a través del IRDS o indirectamente por medio de la característica importar/exportar. El IRD tiene una arquitectura abierta, de modo que las herramientas CASE pueden personalizar y ampliar su esquema conceptual.

Hay dos propuestas primarias para el IRD y el IRDS, que en la actualidad son estándares desarrollados por la International Standards Organization (ISO). Sin embargo, la implementación de estándares no está generalizada. Microsoft y Texas Instruments desarrollaron en forma conjunta el Microsoft Repository, que ofrece soporte para muchos de los objetivos del IRD y el IRDS, aunque no cumple con el estándar. No obstante, Microsoft Repository ha logrado una aceptación generalizada entre los fabricantes de herramientas CASE. En este punto, al parecer, se trata del estándar de facto para el IRD y el IRDS.

**FIGURA 14.7**  
Arquitectura del IRDS



## 14.3 Procesos para especialistas en bases de datos

Esta sección describe los procesos que llevan a cabo los administradores de datos y de bases de datos. Los primeros realizan la planeación de los datos como parte del proceso de planeación de los sistemas de información. Tanto los administradores de datos como los de bases de datos pueden realizar tareas en el proceso de selección y evaluación de los DBMS. Esta sección presenta los detalles de ambos procesos.

### 14.3.1 Planeación de datos

A pesar de las grandes cantidades de dinero invertidas en tecnología de la información, muchas organizaciones se sienten desilusionadas con sus beneficios. Muchas de ellas han creado islas de automatización que ofrecen soporte para los objetivos locales, pero no para las metas globales de la organización. La estrategia de las islas de automatización puede dar lugar a una mala alineación de los objetivos del negocio y de la tecnología de la información. Uno de los resultados de esta alineación equivocada es la dificultad para extraer el valor de la toma de decisiones de las bases de datos operativas.

Como respuesta a los problemas de las islas de automatización, muchas organizaciones llevan a cabo un proceso de planeación detallado para la tecnología y los sistemas de información. El proceso de planeación tiene varios nombres, como planeación de sistemas de información, planeación de sistemas de negocios, ingeniería de sistemas de información y arquitectura de sistemas de información. Todos estos planteamientos ofrecen un proceso para lograr los siguientes objetivos:

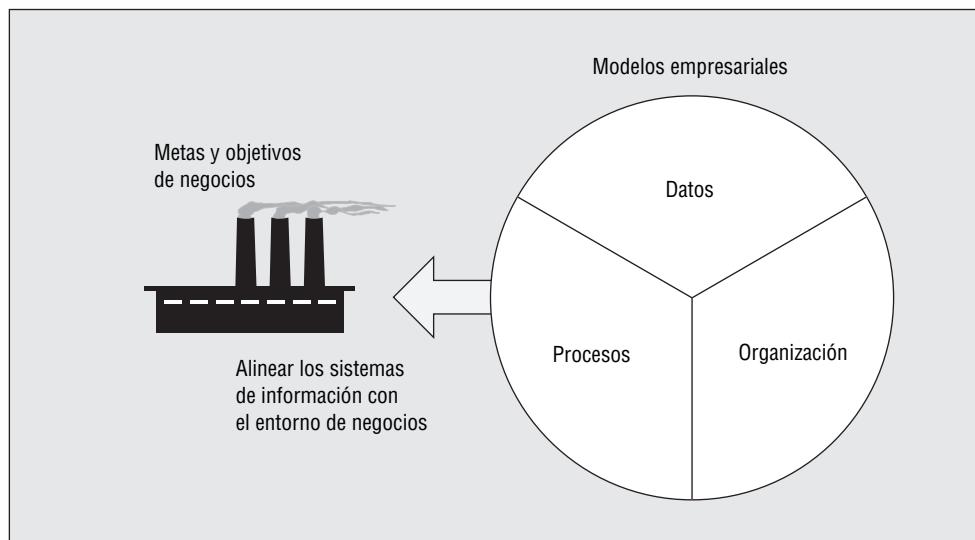
- Evaluación de los sistemas de información actuales en cuanto a las metas y objetivos de la organización.
- Determinación del alcance y lo oportuno que resulta el desarrollo de nuevos sistemas de información, así como el uso de nueva tecnología de la información.
- Identificación de las oportunidades de aplicar la tecnología de la información para lograr una ventaja competitiva.

El proceso de planeación de sistemas de información comprende el desarrollo de los modelos empresariales de datos, procesos y funciones organizacionales, como ilustra la figura 14.8. En la primera parte del proceso de planeación se desarrollan modelos amplios. La tabla 14.11

#### planeación de sistemas de información

el proceso de desarrollar modelos empresariales de datos, procesos y funciones organizacionales. La planeación de sistemas de información evalúa los sistemas existentes, identifica las oportunidades de aplicar la tecnología de la información para lograr una ventaja competitiva y planea nuevos sistemas.

**FIGURA 14.8**  
**Modelos empresariales desarrollados en el proceso de planeación de sistemas de información**



**TABLA 14.11**  
**Nivel de detalle de los modelos empresariales**

Modelo	Niveles de detalle
Datos	Modelo de sujeto (nivel inicial), modelo de entidad (nivel detallado)
Proceso	Áreas funcionales y procesos de negocio (nivel inicial), modelo de actividad (nivel detallado)
Organización	Definiciones de funciones y relaciones de funciones
Interacción de datos y procesos	Matriz y diagramas que muestran los requerimientos de datos del proceso
Interacción de procesos y organización	Matriz y diagramas que muestran las responsabilidades de función
Datos-organización	Matriz y diagramas que muestran el uso de los datos por función

muestra el nivel de detalle inicial para los modelos de datos, procesos y organización. El modelo de datos empresariales se desarrolla primero, ya que casi siempre es más estable que el modelo de procesos. Para integrar estos modelos se desarrollan otros modelos de interacción, como muestra la tabla 14.11. Si se desea obtener detalles adicionales, los modelos de procesos y datos se amplían todavía más. Estos modelos deben reflejar la infraestructura actual de los sistemas de información, así como la dirección planeada para el futuro.

Los administradores de datos desempeñan una función importante en el desarrollo de los planes de sistemas de información: llevan a cabo numerosas entrevistas para desarrollar el modelo de datos empresariales y se coordinan con otros empleados de planeación con el objeto de desarrollar modelos de interacción. Para aumentar las probabilidades de que los planes sean aceptados y utilizados, los administradores de datos deben lograr la participación de la alta gerencia. Si se pone énfasis en el potencial de los sistemas de información integrados para la toma de decisiones, la alta gerencia se sentirá motivada a apoyar el proceso de planeación.

### 14.3.2 Selección y evaluación de los sistemas de administración de bases de datos

La selección y evaluación de un DBMS puede ser una tarea muy importante para una organización. Los DBMS proporcionan una parte importante de la infraestructura de cómputo. Conforme las organizaciones se esfuerzan por manejar el comercio electrónico a través de Internet y extraer el valor de las bases de datos operativas, los DBMS desempeñan una función cada vez más importante. El proceso de selección y evaluación es importante por el impacto que puede tener una mala elección. Los impactos inmediatos pueden ser el lento desempeño de las bases

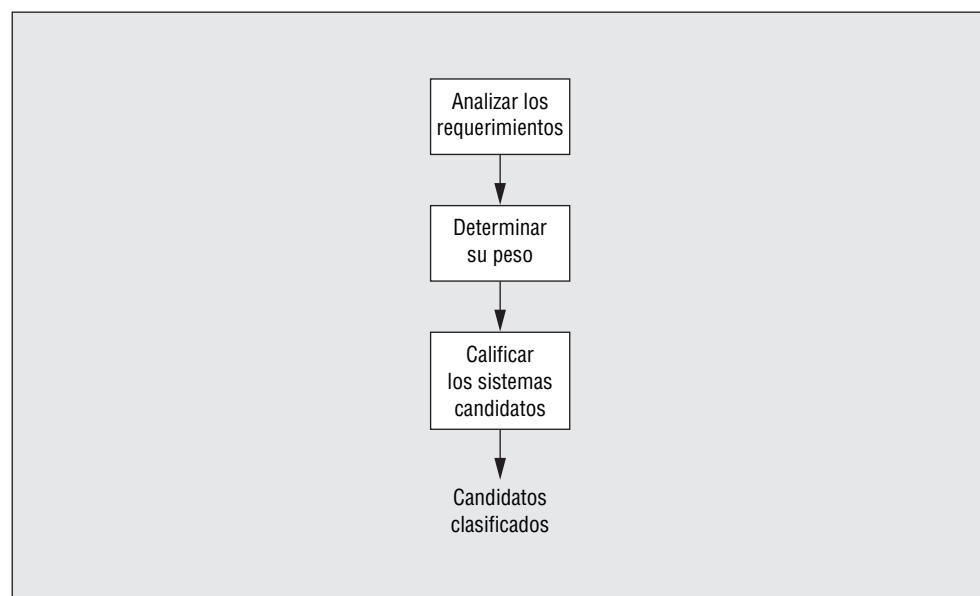
de datos y la pérdida del precio de compra. Un sistema de información de pobre desempeño puede causar pérdidas en ventas y costos más altos. Los impactos a más largo plazo son los elevados costos de cambio. Para cambiar de DBMS es posible que una organización necesite convertir los datos, recodificar el software y volver a capacitar a sus empleados. Los costos del cambio pueden ser mucho más altos que el precio de compra original.

#### *Proceso de selección y evaluación*

El proceso de selección y evaluación comprende una valoración detallada de las necesidades de una organización y las características de los DBMS candidatos. El objetivo del proceso es determinar un grupo reducido de sistemas candidatos que se investigan con mayor detalle. Debido a la naturaleza detallada del proceso, un DBA realiza la mayor parte de las tareas. Por tanto, el DBA necesita conocer bien los DBMS para poder llevarlas a cabo.

La figura 14.9 ilustra los pasos del proceso de selección y evaluación. En el primero, un DBA lleva a cabo un análisis detallado de los requisitos. Como éstos últimos son muchos, resulta útil agruparlos. La tabla 14.12 presenta los principales grupos de requerimientos, mientras que la tabla 14.13 muestra algunos requerimientos individuales en un grupo. Cada uno se debe clasificar como esencial, deseable u opcional para el grupo de requerimientos. En algunos casos se necesitan varios niveles de requisitos. En el caso de los requisitos individuales, un DBA debe ser capaz de medirlos en forma objetiva en los sistemas candidatos.

**FIGURA 14.9**  
Panorama general del proceso de selección y evaluación



**TABLA 14.12**  
Algunos grupos importantes de requerimientos

Categoría
Definición de datos (conceptual)
Recuperación no procedural
Definición de datos (interna)
Desarrollo de aplicaciones
Lenguaje procedural
Control de concurrencia
Manejo de la recuperación
Procesamiento y datos distribuidos
Soporte para fabricantes
Optimización de consultas

**TABLA 14.13**

**Algunos requerimientos detallados para la categoría de definición de datos conceptuales**

Requisito (importancia)	Explicación
Integridad de la entidad (esencial)	Declaración y aplicación de llaves primarias
Llaves candidatas (deseables)	Declaración y aplicación de llaves candidatas
Integridad referencial (esencial)	Declaración y aplicación de la integridad referencial
Filas referenciadas (deseables)	Declaración y aplicación de las reglas para las filas referenciadas
Tipos de datos estándar (esenciales)	Soporte para números enteros (varios tamaños), números con punto flotante (varios tamaños), números con punto fijo, cadenas de longitud fija, cadenas de longitud variable y datos (fecha, hora y sello de hora)
Tipos de datos definidos por el usuario (deseables)	Soporte para nuevos tipos de datos o un menú de tipos de datos opcionales
Interfaz de usuario (deseable)	Interfaz gráfica de usuario para ampliar los enunciados sentencias CREATE de SQL
Afirmaciones generales (opcionales)	Declaración y aplicación de limitaciones para tablas múltiples

**TABLA 14.14**

**Interpretación de los valores de calificación para las comparaciones pareadas**

Valor de calificación de $A_{ij}$	Significado
1	Los requerimientos $i$ y $j$ tienen la misma importancia.
3	El requerimiento $i$ es ligeramente más importante que el requerimiento $j$ .
5	El requerimiento $i$ es significativamente más importante que el requerimiento $j$ .
7	El requerimiento $i$ es mucho más importante que el requerimiento $j$ .
9	El requerimiento $i$ es en extremo más importante que el requerimiento $j$ .

**TABLA 14.15**

**Ponderaciones de muestra para algunos grupos de requerimientos**

	Definición de datos (Conceptual)	Recuperación no procedural	Desarrollo de aplicaciones	Control de concurrencia
Definición de datos (conceptual)	1	1/5 (0.20)	1/3 (0.33)	1/7 (0.14)
Recuperación no procedural	5	1	3	1/3 (0.33)
Desarrollo de aplicaciones	3	1/3 (0.33)	1	1/5 (0.20)
Control de concurrencia	7	3	5	1
Suma de columnas	16	4.53	9.33	1.67

### proceso jerárquico analítico

una técnica de la teoría de las decisiones para evaluar los problemas con varios objetivos. El proceso se puede utilizar para seleccionar y evaluar DBMS permitiendo una asignación sistemática de ponderaciones a los requerimientos y calificaciones a las características de los DBMS candidatos.

Después de determinar los grupos, el DBA debe asignar las ponderaciones a los principales grupos de requerimientos y calificar los sistemas candidatos. Con una mayor cantidad de grupos de requerimientos puede ser muy difícil asignar ponderaciones consistentes. El DBA necesita una herramienta que le ayude a asignar ponderaciones consistentes y a calificar los sistemas candidatos. Por desgracia, ningún método analítico para asignación de ponderaciones y calificación de sistemas ha logrado un uso generalizado. Con el objeto de fomentar el uso de métodos analíticos para asignación de ponderaciones y calificación, ilustramos una estrategia prometedora.

El proceso jerárquico analítico ofrece un planteamiento sencillo que alcanza un nivel de consistencia razonable. Utilizando este proceso, un DBA asigna ponderaciones a las combinaciones pareadas de grupos de requisitos. Por ejemplo, un DBA debe asignar una ponderación que represente la importancia de la definición conceptual de los datos, en comparación con la recuperación no procedural. El proceso jerárquico analítico proporciona una escala de nueve puntos con las interpretaciones que muestra la tabla 14.14. La tabla 14.15 aplica la escala para calificar algunos de los grupos de requerimientos en la tabla 14.12. Para la consistencia, si se

**TABLA 14.16**  
Ponderaciones normalizadas para algunos grupos de requisitos

	Definición de datos (Conceptual)	Recuperación no procedural	Desarrollo de aplicaciones	Control de concurrencia
Definición de datos (conceptual)	0.06	0.04	0.04	0.08
Recuperación no procedural	0.31	0.22	0.32	0.20
Desarrollo de aplicaciones	0.19	0.07	0.11	0.12
Control de concurrencia	0.44	0.66	0.54	0.60

**TABLA 14.17**  
Valores de importancia para algunos grupos de requerimientos

Grupo de requerimientos	Importancia
Definición de datos (conceptual)	0.06
Recuperación no procedural	0.26
Desarrollo de aplicaciones	0.12
Control de concurrencia	0.56

captura  $A_{ij} = x$ , entonces,  $A_{ji} = 1/x$ . Además, los elementos en diagonal de la tabla 14.15 siempre deben ser 1. Por tanto, es necesario completar sólo la mitad de las calificaciones en la tabla 14.15. El último renglón de la matriz muestra las sumas de las columnas utilizadas para normalizar las ponderaciones y determinar los valores de importancia.

Después de asignar ponderaciones pareadas a los grupos de requisitos, éstas se combinan para determinar la importancia de cada grupo de requerimientos. Los valores de las celdas se normalizan dividiendo cada celda entre la suma de sus columnas, como se muestra en la tabla 14.16. El valor de importancia final de cada grupo de requerimientos es el promedio de las ponderaciones normalizadas en cada fila, como ilustra la tabla 14.17.

Es necesario calcular las ponderaciones de importancia para cada subcategoría de los grupos de requerimientos de la misma manera que para los grupos de requerimientos. Para cada subcategoría, las ponderaciones pareadas se asignan antes de normalizar las ponderaciones y calcular los valores de importancia final.

Después de calcular los valores de importancia para los requerimientos, se asignan las calificaciones a los DBMS candidatos. Calificar a los DBMS candidatos puede ser difícil por el número de requerimientos individuales y la necesidad de combinar estos requerimientos en una calificación general para el grupo. Como primera parte del proceso de calificación, un DBA debe investigar con detenimiento las características de cada DBMS candidato.

Se han propuesto muchos planteamientos para combinar las calificaciones de las características individuales en una calificación general para el grupo de requerimientos. El proceso analítico jerárquico ofrece soporte para comparaciones pareadas entre DBMS candidatos utilizando los valores de calificación de la tabla 14.14. Las interpretaciones cambian ligeramente para reflejar las comparaciones entre los DBMS candidatos en lugar de la importancia de los grupos de requerimientos. Por ejemplo, es preciso asignar un valor 3 si el DBMS  $i$  es ligeramente mejor que el DBMS  $j$ . Para cada subcategoría de requerimientos, es necesario crear una matriz de comparación para comparar los DBMS candidatos. Las calificaciones para cada DBMS se calculan normalizando las ponderaciones y calculando los promedios de las filas, como en el caso de los grupos de requerimientos.

Después de calificar los DBMS candidatos para cada grupo de requerimientos, las calificaciones finales se calculan combinando las calificaciones y la importancia de los grupos de requerimientos. Para más detalles sobre el cálculo de las calificaciones finales, tiene que consultar las referencias al final del capítulo sobre el proceso analítico jerárquico.

**evaluación comparativa (*benchmark*)**  
una carga de trabajo para evaluar el desempeño de un sistema o producto. Una buena evaluación comparativa debe ser relevante, compatible, escalable y entendible.

### Proceso de selección final

Después de terminar el proceso de selección y evaluación, es necesario evaluar con más detalle los dos o tres DBMS candidatos ganadores. Es posible utilizar evaluaciones comparativas para proporcionar una valoración más detallada de estos candidatos. Una evaluación comparativa (*benchmark*) es una carga de trabajo para evaluar el desempeño de un sistema o producto. Una buena evaluación comparativa debe ser relevante, compatible, escalable y entendible. Como el desarrollo de evaluaciones comparativas eficientes requiere de mucha experiencia, la mayoría de las organizaciones no deben tratar de desarrollar una evaluación de este tipo. Por suerte, el Transaction Processing Council (TPC) ha desarrollado gran cantidad de evaluaciones comparativas estándar y específicas para dominio, como resume la tabla 14.18. Cada evaluación comparativa se desarrolló durante mucho tiempo con información de un grupo de contribuidores muy diverso.

Un DBA puede utilizar los resultados del TPC para cada evaluación comparativa y así obtener estimados razonables acerca del desempeño de un DBMS en particular en un entorno de hardware/software específico. Los resultados del desempeño del TPC comprenden el desempeño total del sistema y no sólo el desempeño del DBMS, de modo que los resultados no se exageran cuando un cliente usa un DBMS en un entorno específico de hardware/software. Para facilitar las comparaciones entre el precio y el desempeño, el TPC publica la medida del desempeño además del precio/desempeño para cada evaluación comparativa. El precio cubre todas las dimensiones de costos de un entorno completo de sistemas, incluidas estaciones de trabajo, equipo de comunicación, software de sistemas, sistema de cómputo o *host*, almacenamiento de respaldos y costo de mantenimiento por tres años. El TPC realiza una auditoría de los resultados de las evaluaciones comparativas antes de su publicación para asegurarse de que los fabricantes no manipularon los resultados.

Para mejorar los resultados publicados del TPC, quizás una organización quiera evaluar un DBMS por medio de ensayos. Es posible crear evaluaciones comparativas personalizadas para medir la eficiencia de un DBMS en cuanto al uso para el que está diseñado. Además, la interfaz de usuario y las capacidades de desarrollo de aplicaciones se pueden evaluar mediante la creación de pequeñas aplicaciones.

La etapa final del proceso de selección puede comprender consideraciones no técnicas que presentan los administradores de datos en conjunto con la alta gerencia y el personal legal. La evaluación de los prospectos futuros por parte de cada fabricante es importante porque los sistemas de información pueden durar mucho tiempo. Si el DBMS subyacente no avanza con la industria, es probable que no ofrezca soporte para iniciativas futuras y actualizaciones a los sistemas de información que las emplean. Debido a los altos costos fijos y variables (cuotas de mantenimiento) de un DBMS, a menudo la negociación es un elemento crucial del proceso de selección final. Los términos del contrato final, además de una o dos ventajas clave, a menudo marcan la diferencia en la selección final.

El software de DBMS de código abierto es un desarrollo reciente que complica el proceso de selección y evaluación. Este tipo de software genera incertidumbre en cuanto a las licencias y los prospectos futuros, pero también ofrece ventajas obvias en cuanto al precio de compra sobre el software comercial. Con el software gratuito, la falta de incentivos por utilidades podría afectar las actualizaciones al producto y llevar a cambios en las licencias de software para

**TABLA 14.18**  
**Resumen de las evaluaciones comparativas del TPC**

Evaluación comparativa ( <i>benchmark</i> )	Descripción	Medidas de desempeño
TPC-C	Evaluación comparativa de la captura de pedidos en línea	Transacciones por minuto; precio por transacciones por minuto
TPC-H	Soporte para consultas <i>ad hoc</i>	Consultas compuestas por hora; precio de peticiones compuestas por hora
TPC-App	Procesamiento de transacciones de negocio a negocio con servicios de aplicaciones y web	Interacciones de los servicios web por segundo (SIPS) por servidor de aplicaciones; SIPS totales; precio por SIPS
TPC-W	Evaluación comparativa del comercio en línea	Interacciones web por segundo; precio por interacciones web por segundo

obtener actualizaciones de los productos. Por ejemplo, MySQL, el DBMS gratuito más popular, hace poco cambió sus licencias, de modo que los usuarios comerciales van a tener que pagar una tarifa por ellas. A pesar de estas incertidumbres, muchas organizaciones utilizan el software DBMS gratuito sobre todo para sistemas que no son de misión crítica.

## 14.4 Administración de entornos de bases de datos

Los DBMS funcionan en distintos entornos de procesamiento. Los especialistas en datos deben entender los entornos para garantizar un desempeño adecuado de las bases de datos y establecer estándares y políticas. Esta sección ofrece un panorama general de los entornos de procesamiento enfatizando las tareas que realizan los administradores de bases de datos y de datos. Los demás capítulos de la parte 4 proporcionan los detalles de los entornos de procesamiento.

### 14.4.1 Procesamiento de transacciones

El procesamiento de transacciones comprende las operaciones cotidianas de una organización. Todos los días, las organizaciones procesan grandes volúmenes de pedidos, pagos, retiros de efectivo, reservaciones de líneas aéreas, cobros de seguros y otros tipos de transacciones. Los DBM prestan los servicios esenciales para realizar las operaciones de manera eficiente y confiable. Organizaciones como los bancos con los cajeros automáticos, las líneas aéreas con los sistemas de reservaciones en línea y universidades con registros en línea, no podrían funcionar sin un procesamiento de operaciones confiable y eficiente. Con el interés cada vez mayor de hacer negocios a través de Internet, el procesamiento de transacciones adquiere mayor importancia día a día.

Como muestra la tabla 14.19, los especialistas en datos tienen muchas responsabilidades en cuanto al procesamiento de transacciones. Los administradores de datos tienen responsabilidades de planeación que comprenden la infraestructura y la recuperación en caso de desastre. Los administradores de bases de datos casi siempre realizan las tareas más detalladas, como consultoría sobre el diseño de operaciones y supervisión del desempeño. En vista de la importancia del procesamiento de transacciones, a menudo los administradores de bases de datos deben estar al pendiente para solucionar los problemas. El capítulo 15 presenta los detalles del procesamiento de transacciones para el control de concurrencia y la administración de la recuperación. Después de leer el capítulo 15, tal vez quiera revisar otra vez la tabla 14.19.

### 14.4.2 Procesamiento de *data warehouses*

El *data warehousing* comprende el soporte de las decisiones de bases de datos. Como muchas organizaciones no han podido utilizar las bases de datos operativas directamente para ofrecer soporte en la toma de decisiones administrativas, surgió la idea de un *data warehouse*. Un *data warehouse* es una base de datos central en la que se guardan los datos de toda la empresa para facilitar las actividades de soporte de decisiones por parte de los departamentos usuarios. La información de las bases de datos operativas y de fuentes externas se extrae, limpia, integra, y después se carga en un *data warehouse*. Como un *data warehouse* contiene datos históricos, la mayor parte de la actividad consiste en la recuperación de información resumida.

**TABLA 14.19**  
Responsabilidades  
de los especialistas en  
bases de datos para  
el procesamiento de  
transacciones

Área	Responsabilidades
Diseño de transacciones	Consultar sobre el diseño para equilibrar la integridad y el desempeño; educar sobre los aspectos de diseño y las características de los DBMS
Monitoreo del desempeño	Monitorear el desempeño de las transacciones y solucionar los problemas de desempeño; modificar los niveles de los recursos para mejorar el desempeño
Infraestructura para el procesamiento de transacciones	Determinar los niveles de los recursos en cuanto a su eficiencia (disco, memoria y CPU) y confiabilidad (nivel RAID)
Recuperación de desastres	Proporcionar planes de contingencia para distintos tipos de fallas en las bases de datos

Como muestra la tabla 14.20, los especialistas en datos tienen muchas responsabilidades en los *data warehouses*. Los administradores de datos pueden tener responsabilidades que comprenden la arquitectura de los *data warehouses* y el modelo de datos empresariales. Por lo general, los administradores de bases de datos realizan las tareas más detalladas, como el monitoreo del desempeño y la consultoría. Con el fin de ofrecer soporte para un *data warehouse* extenso, quizás sean necesarios otros productos de software además de un DBMS. Es necesario llevar a cabo un proceso de selección y evaluación para elegir el producto más apropiado. El capítulo 16 presenta los detalles de los *data warehouse*. Después de leer el capítulo 16, tal vez quiera revisar otra vez la tabla 14.20.

#### 14.4.3 Entornos distribuidos

Los DBMS pueden operar en entornos distribuidos para soportar tanto el procesamiento de transacciones como los *data warehouse*. En los entornos distribuidos, los DBMS ofrecen la habilidad de distribuir el procesamiento y los datos entre computadoras conectadas en red. Para el procesamiento distribuido, un DBMS puede permitir la distribución entre varias computadoras en una red de las funciones que ofrece el DBMS, así como el procesamiento de aplicaciones a distribuir. Para los datos distribuidos, un DBMS puede permitir que las tablas se guarden y quizás se repliquen en distintas computadoras de una red. La capacidad de distribuir el procesamiento y los datos promete mayor flexibilidad, escalabilidad, desempeño y confiabilidad. Sin embargo, estas mejoras sólo se pueden lograr a través de un diseño detallado.

Como muestra la tabla 14.21, en los entornos de bases de datos distribuidas, los especialistas en datos tienen muchas responsabilidades. Los administradores de datos tienen responsabilidades de planeación que comprenden establecer metas y determinar arquitecturas. Como los entornos distribuidos no aumentan la funcionalidad, es preciso justificarlos con las mejoras en las aplicaciones subyacentes. Los administradores de bases de datos realizan tareas más detalladas,

**TABLA 14.20**  
Responsabilidades de los especialistas en bases de datos en los *data warehouses*

Área	Responsabilidades
Uso de los <i>data warehouse</i>	Educar y consultar sobre el diseño de aplicaciones y las características de los DBMS para el procesamiento de <i>data warehouse</i>
Monitoreo del desempeño	Monitorear el desempeño de los <i>data warehouse</i> y solucionar los problemas de integridad; modificar los niveles de recursos para mejorar el desempeño
Actualización del <i>data warehouse</i>	Determinar la frecuencia para actualizar el <i>data warehouse</i> y programar las actividades para actualizarlo
Arquitectura del <i>data warehouse</i>	Determinar la arquitectura para soportar las necesidades de toma de decisiones; seleccionar productos de bases de datos para soportar la arquitectura; determinar los niveles para un procesamiento eficiente
Modelo de datos empresariales	Ofrecer su experiencia en el contenido de las bases de datos operativas; determinar el modelo de datos conceptuales para el <i>data warehouse</i> ; promover la calidad de los datos para soportar el desarrollo del <i>data warehouse</i>

**TABLA 14.21**  
Responsabilidades de los especialistas en bases de datos para entornos distribuidos

Área	Responsabilidades
Desarrollo de aplicaciones	Educar y consultar sobre el impacto de los ambientes distribuidos para el procesamiento de transacciones y los <i>data warehouse</i>
Monitoreo del desempeño	Monitorear el desempeño y solucionar los problemas con un énfasis especial en el ambiente distribuido
Arquitecturas de entornos distribuidos	Identificar las metas para los entornos distribuidos; elegir el procesamiento distribuido, las bases de datos paralelas y las arquitecturas de bases de datos distribuidas para lograr los objetivos; seleccionar productos de software adicionales que soporten las arquitecturas
Diseño de entornos distribuidos	Diseñar bases de datos distribuidas; determinar los niveles de recursos para un procesamiento eficiente

**TABLA 14.22**  
**Responsabilidades de los especialistas en bases de datos en las bases de datos de objetos**

Área	Responsabilidades
Desarrollo de aplicaciones	Educar y consultar sobre la creación de nuevos tipos de datos, la herencia de tipos de datos y tablas, y otras características de los objetos
Monitoreo del desempeño	Monitorear el desempeño y solucionar los problemas con nuevos tipos de datos
Arquitecturas de bases de datos de objetos	Identificar las metas para los DBMS de objetos; elegir las arquitecturas de bases de datos de objetos
Diseño de bases de datos de objetos	Diseñar bases de datos de objetos; seleccionar tipos de datos; crear nuevos tipos de datos

como el monitoreo del desempeño y el diseño de bases de datos distribuidas. Para soportar entornos distribuidos, quizás sean necesarios otros productos de software además de las extensiones principales de un DBMS. Es preciso realizar un proceso de selección y evaluación para elegir los productos más apropiados. El capítulo 17 presenta los detalles del procesamiento y los datos distribuidos. Después de leer el capítulo 17, tal vez quiera revisar otra vez la tabla 14.21.

#### 14.4.4 Administración de bases de datos de objetos

Los DBMS de objetos soportan una funcionalidad adicional para el procesamiento de transacciones y las aplicaciones de *data warehouses*. Muchos sistemas de información utilizan mayor variedad de tipos de datos que la que ofrecen los DBMS relacionales. Por ejemplo, muchas bases de datos financieras necesitan manipular series de tiempo, un tipo de datos que no ofrecen la mayor parte de los DBMS relacionales. Con la capacidad de convertir cualquier tipo de datos en formato digital, la necesidad de nuevos tipos de datos es todavía más evidente. A menudo, las bases de datos de negocios necesitan integrar tanto datos tradicionales con datos no tradicionales basados en nuevos tipos. Por ejemplo, los sistemas de información para procesar cobros de seguros deben manejar datos tradicionales como imágenes, mapas y dibujos. En vista de estas necesidades, los DBMS de relación existentes se han ampliado con capacidades de objetos y se han desarrollado nuevos DBMS de objetos.

Como indica la tabla 14.22, los especialistas en datos tienen muchas responsabilidades en cuanto a las bases de datos de objetos. Los administradores de datos generalmente llevan a cabo la planeación de responsabilidades que incluyen las metas de configuración y la determinación de arquitecturas. Por lo regular, los administradores de bases de datos realizan tareas más detalladas como monitoreo del desempeño, consultoría y diseño de bases de datos de objetos. Un DBMS de objetos puede ser una extensión importante de un DBMS relacional existente o un DBMS nuevo. Es necesario llevar a cabo un proceso de selección y evaluación para elegir el producto más apropiado. El capítulo 18 presenta los detalles de los DBMS de objetos. Después de leer el capítulo 18, es probable que quiera volver a revisar la tabla 14.22.

---

#### Reflexión final

Este capítulo describió las responsabilidades, herramientas y procesos que utilizan los especialistas en bases de datos y que soportan la toma de decisiones. Muchas organizaciones proporcionan dos funciones para administrar los recursos de información. Los administradores de datos llevan a cabo una planeación extensa y establecen políticas, mientras que los administradores de bases de datos vigilan con detenimiento las bases de datos individuales y los DBMS. Con el fin de ofrecer un contexto para entender las responsabilidades de estos puestos, este capítulo estudió la filosofía de la administración de los recursos de información que enfatiza la tecnología de la información como una herramienta para procesar, distribuir e integrar la información en toda una organización.

Este capítulo describió varias herramientas que apoyan a los administradores de bases de datos, quienes utilizan reglas de seguridad para restringir el acceso, así como limitaciones de integridad para mejorar la calidad de los datos. Este capítulo describió las reglas de seguridad y las limitaciones de integridad, así como la sintaxis SQL:2003 asociada. En cuanto a los disparadores

y procedimientos almacenados, este capítulo describió las responsabilidades administrativas de los DBA para complementar los detalles de encriptación en el capítulo 11. El diccionario de datos es una herramienta importante para administrar bases de datos individuales, así como para integrar el desarrollo de bases de datos con el desarrollo de sistemas de información. Este capítulo presentó dos tipos de diccionarios de datos: tablas de catálogos que utilizan los DBMS y el diccionario de recursos de información que usan las herramientas CASE.

Los especialistas en bases de datos necesitan entender dos importantes procesos para manejar la tecnología de información. Los administradores de datos participan en un proceso de planeación detallado que determina nuevas direcciones para el desarrollo de sistemas de información. Este capítulo describió el proceso de planeación de datos como un componente importante del proceso de planeación de sistemas de información. Tanto los administradores de datos como los administradores de bases de datos participan en la selección y evaluación de los DBMS. Los segundos realizan las tareas detalladas, mientras que los administradores de datos a menudo toman las decisiones finales de selección utilizando recomendaciones detalladas. Este capítulo describió los pasos del proceso de selección y evaluación y las tareas que realizan los administradores de datos y de bases de datos en el proceso.

Este capítulo ofrece un contexto para los otros capítulos de la parte 7. Los demás capítulos proporcionan detalles sobre distintos entornos de bases de datos, que incluyen procesamiento de transacciones, *data warehouses*, entornos distribuidos y DBMS de objetos. Este capítulo enfatiza las responsabilidades, herramientas y procesos de los especialistas en bases de datos para administrar dichos entornos. Después de completar los otros capítulos de la parte 7, le sugerimos volver a leer éste para integrar los detalles con los conceptos y las técnicas administrativas.

## Revisión de conceptos

- Administración de recursos de información: filosofía administrativa para controlar los recursos de información y aplicar la tecnología de información con el fin de soportar la toma de decisiones administrativas.
- Administrador de bases de datos: puesto de apoyo para administrar bases de datos individuales y DBMS.
- Administrador de datos: puesto administrativo con responsabilidades de planeación y políticas para la tecnología de la información.
- Controles de acceso discretionarios para asignar derechos de acceso a grupos y usuarios.
- Controles de acceso obligatorios para bases de datos muy delicadas y estáticas utilizadas en la recopilación de inteligencia y la defensa nacional.
- Sentencias SQL CREATE/DROP ROLE y sentencias GRANT/REVOKE para reglas de autorización discrecional.

**CREATE ROLE ISFaculty**

**GRANT SELECT ON ISSStudentGPA  
TO ISFaculty, ISAdvisor, ISAdministrator**

**REVOKE SELECT ON ISSStudentGPA FROM ISFaculty**

- Sistema Oracle 10g y privilegios de objetos para el control de acceso discrecional.
- Sentencia SQL CREATE DOMAIN para limitaciones de tipos de datos.

**CREATE DOMAIN StudentClass AS CHAR(2)  
CHECK (VALUE IN ('FR', 'SO', 'JR', 'SR'))**

- Tipos SQL distintivos para mejorar la verificación de tipos.
- Limitaciones de dominios SQL y tipos distintivos comparados con tipos de datos definidos por el usuario.
- Sentencia SQL CREATE ASSERTION para limitaciones de integridad complejas.

**CREATE ASSERTION OfferingConflict  
CHECK (NOT EXISTS**

```
( SELECT O1.OfferNo
    FROM Offering O1, Offering O2
   WHERE O1.OfferNo <> O2.OfferNo
     AND O1.OffTerm = O2.OffTerm
     AND O1.OffYear = O2.OffYear
     AND O1.OffDays = O2.OffDays
     AND O1.OffTime = O2.OffTime
     AND O1.OffLocation = O2.OffLocation ) )
```

- Limitación CHECK en la sentencia CREATE TABLE para limitaciones que comprenden las condiciones de fila en columnas de la misma tabla.

```
CREATE TABLE Student
( StdSSN      CHAR(11),
  StdFirstName VARCHAR(50)  CONSTRAINT StdFirstNameRequired
                           NOT NULL,
  StdLastName  VARCHAR(50)  CONSTRAINT StdLastNameRequired
                           NOT NULL,
  StdCity      VARCHAR(50)  CONSTRAINT StdCityRequired NOT
                           NULL,
  StdState     CHAR(2)  CONSTRAINT StdStateRequired NOT
                           NULL,
  StdZip       CHAR(9)  CONSTRAINT StdZipRequired NOT
                           NULL,
  StdMajor     CHAR(6),
  StdClass     CHAR(6),
  StdGPA       DECIMAL(3,2),
CONSTRAINT PKStudent PRIMARY KEY (StdSSN),
CONSTRAINT ValidGPA CHECK ( StdGPA BETWEEN 0 AND 4 ),
CONSTRAINT MajorDeclared CHECK
( StdClass IN ('FR', 'SO') OR StdMajor IS NOT NULL ) )
```

- Administración de prácticas de encriptación de disparadores y procedimientos: estándares de documentación, uso de parámetros y contenido.
- Administración de dependencias de objetos: obsolescencia de planes de acceso, modificación de los objetos referenciados, eliminación de los objetos referenciados.
- Control de la complejidad de los disparadores: identificar las interacciones de los disparadores, minimizar las acciones de los disparadores que pueden activar otros disparadores, sin dependencia de un orden de activación específico para los disparadores superpuestos.
- Tablas de catálogos para rastrear los objetos que maneja un DBMS.
- Diccionario de recursos de información para la administración del proceso de desarrollo de sistemas de la información.
- Desarrollo de un modelo de datos empresariales como parte importante del proceso de planeación de los sistemas de información.
- Proceso de selección y evaluación para analizar las necesidades organizacionales y las características de los DBMS.
- Uso de una herramienta como el proceso jerárquico analítico para asignar en forma consistente las ponderaciones de importancia y calificar a los DBMS candidatos.
- Uso de los resultados de las evaluaciones comparativas de dominios estándar para medir el desempeño de los DBMS.
- Responsabilidades de los especialistas en bases de datos para administrar el procesamiento de transacciones, los *data warehouse*, los entornos distribuidos y los DBMS de objetos.

## Preguntas

1. ¿Por qué es difícil utilizar bases de datos operativas para la toma de decisiones administrativas?
2. ¿De qué manera se deben transformar las bases de datos para la toma de decisiones administrativas?
3. ¿Cuáles son las etapas del ciclo de vida de la información?
4. ¿Qué significa integrar los ciclos de vida de la información?
5. ¿Qué dimensión de la calidad de los datos es importante para la toma de decisiones administrativas, pero no para la toma de decisiones operativas?
6. ¿En qué se diferencian la administración del conocimiento y la administración de recursos de información?
7. ¿Cuáles son los tres pilares de la administración del conocimiento?
8. ¿Qué tipo de puesto es el de administrador de datos?
9. ¿Qué tipo de puesto es el de administrador de bases de datos?
10. ¿Qué puesto (administrador de datos o de bases de datos) tiene un panorama más amplio de los recursos de información?
11. ¿Qué es un modelo de datos empresariales?
12. ¿Para qué se desarrolla un modelo de datos empresariales?
13. ¿Qué clase de especializaciones son posibles en las grandes organizaciones para los administradores de datos y de bases de datos?
14. ¿Qué es control de acceso discrecional?
15. ¿Qué es control de acceso obligatorio?
16. ¿Qué tipo de base de datos requiere de un control de acceso obligatorio?
17. ¿Cuáles son los propósitos de las sentencias GRANT y REVOKE en SQL?
18. ¿Por qué las reglas de autorización deben hacer referencia a las funciones en lugar de los usuarios individuales?
19. ¿Por qué las reglas de autorización casi siempre utilizan vistas en lugar de tablas o columnas?
20. ¿Cuáles son los dos usos de la sentencia GRANT?
21. ¿Por qué un DBA debe utilizar con precaución la cláusula WITH ADMIN en la sentencia CREATE ROLE y la cláusula WITH GRANT OPTION en la sentencia GRANT?
22. ¿Cuál es la diferencia entre privilegios de sistema y privilegios de objeto en Oracle? Mencione el ejemplo de un privilegio de sistema y un privilegio de objeto.
23. ¿Qué otras disciplinas comprende la seguridad para computadoras?
24. ¿Cuál es el propósito de la sentencia CREATE DOMAIN? Compare y contraste un dominio SQL con un tipo distintivo.
25. ¿Qué capacidades adicionales agrega SQL:2003 para los tipos definidos por el usuario en comparación con los dominios?
26. ¿Cuál es el propósito de las afirmaciones en SQL?
27. ¿Qué significa decir que una afirmación es diferible?
28. ¿Qué alternativas hay para las afirmaciones SQL? ¿Qué objeto tiene utilizar una alternativa para una afirmación?
29. ¿Cuáles son los aspectos de la encriptación de los que un DBA se debe preocupar?
30. ¿De qué manera un disparador o procedimiento almacenado depende de otros objetos de la base de datos?
31. ¿Cuáles son las responsabilidades de un DBA en el manejo de dependencias?
32. ¿Cuál es la diferencia entre el mantenimiento de una dependencia de sello de tiempo y firma?
33. Mencione por lo menos tres formas en las que un DBA puede controlar las interacciones de los disparadores.
34. ¿Qué tipo de metadatos contiene un diccionario de datos?
35. ¿Cuáles son las tablas de catálogos? ¿Qué clase de tablas de catálogos administran los DBMS?
36. ¿Cuál es la diferencia entre el Information\_Schema y el Definition\_Schema en SQL:2003?
37. ¿Por qué es necesario aprender las tablas de catálogos de un DBMS específico?
38. ¿De qué manera un DBA tiene acceso a las tablas de catálogos?
39. ¿Cuál es el propósito de un diccionario de recursos de información?
40. ¿Qué funciones realiza un sistema de diccionarios de recursos de información?

41. ¿Cuáles son los propósitos de la planeación de sistemas de información?
42. ¿Por qué el modelo de datos empresariales se desarrolla antes que el modelo de procesos?
43. ¿Por qué el proceso de selección y evaluación es importante para los DBMS?
44. ¿Cuáles son algunas de las dificultades en el proceso de selección y evaluación para un producto complejo como un DBMS?
45. ¿Cuáles son los pasos del proceso de selección y evaluación?
46. ¿Cómo se utiliza el proceso jerárquico analítico en el proceso de selección y evaluación?
47. ¿Qué responsabilidades tiene el administrador de bases de datos en el proceso de selección y evaluación?
48. ¿Qué responsabilidades tiene el administrador de datos en el proceso de selección y evaluación?
49. ¿Cuáles son las responsabilidades de los administradores de bases de datos en cuanto al procesamiento de transacciones?
50. ¿Cuáles son las responsabilidades de los administradores de bases de datos en cuanto a la administración de *data warehouses*?
51. ¿Cuáles son las responsabilidades de los administradores de bases de datos en cuanto a la administración de bases de datos en entornos distribuidos?
52. ¿Cuáles son las responsabilidades de los administradores de bases de datos en cuanto a la administración de bases de datos de objetos?
53. ¿Cuáles son las responsabilidades de los administradores de datos en cuanto al procesamiento de transacciones?
54. ¿Cuáles son las responsabilidades de los administradores de datos en cuanto a la administración de *data warehouses*?
55. ¿Cuáles son las responsabilidades de los administradores de datos en cuanto a la administración de bases de datos en ambientes distribuidos?
56. ¿Cuáles son las responsabilidades de los administradores de datos en cuanto a la administración de bases de datos de objetos?
57. ¿Cuáles son las características de una buena evaluación comparativa?
58. ¿Por qué el Transaction Processing Council publica medidas del desempeño de los sistemas completos, en lugar de medidas de los componentes?
59. ¿Por qué el Transaction Processing Council publica resultados de precio/desempeño?
60. ¿De qué manera el Transaction Processing Council garantiza que los resultados de las evaluaciones comparativas son relevantes y confiables?

## Problemas

Debido a la naturaleza de este capítulo, los problemas son más abiertos que los de otros capítulos. Al final del resto de los capítulos de la parte 7 aparecen problemas más detallados.

1. Elabore una breve presentación (6 a 12 diapositivas) sobre la evaluación comparativa TPC-C. Deberá proporcionar detalles sobre su historia, el diseño de las bases de datos, detalles de las aplicaciones y resultados recientes.
2. Elabore una breve presentación (6 a 12 diapositivas) sobre la evaluación comparativa TPC-H. Deberá proporcionar detalles sobre su historia, el diseño de las bases de datos, detalles de las aplicaciones y resultados recientes.
3. Elabore una breve presentación (6 a 12 diapositivas) sobre la evaluación comparativa TPC-W. Deberá proporcionar detalles sobre su historia, el diseño de las bases de datos, detalles de las aplicaciones y resultados recientes.
4. Elabore una breve presentación (6 a 12 diapositivas) sobre la evaluación comparativa TPC-App. Deberá proporcionar detalles sobre su historia, el diseño de las bases de datos, detalles de las aplicaciones y resultados recientes.
5. Compare y contraste las licencias de software para MySQL y otro producto DBMS de código abierto.
6. Elabore una lista detallada de requerimientos para la recuperación no procedural. Utilice la tabla 14.13 como lineamiento.
7. Proporcione ponderaciones de importancia a su lista de requerimientos detallados del problema 6 utilizando los criterios AHP de la tabla 14.4.

8. Normalice las ponderaciones y calcule los valores de importancia para sus requerimientos detallados utilizando las ponderaciones de importancia del problema 7.
9. Escriba limitaciones CHECK con nombre para las siguientes reglas de integridad. Modifique la sentencia CREATE TABLE para agregar las limitaciones CHECK nombradas.

```
CREATE TABLE Customer
( CustNo           CHAR(8),
  CustFirstName   VARCHAR2(20) CONSTRAINT CustFirstNameRequired
                   NOT NULL,
  CustLastName    VARCHAR2(30) CONSTRAINT CustLastNameRequired
                   NOT NULL,
  CustStreet      VARCHAR2(50),
  CustCity        VARCHAR2(30),
  CustState       CHAR(2),
  CustZip         CHAR(10),
  CustBal         DECIMAL(12,2) DEFAULT 0,
  CONSTRAINT PKCustomer PRIMARY KEY (CustNo) )
```

- El saldo del cliente es mayor o igual que 0.
  - El estado del cliente puede ser CO, CA, WA, AZ, UT, NV, ID u OR.
10. Escriba limitaciones con el nombre CHECK para las siguientes reglas de integridad. Modifique la sentencia CREATE TABLE para agregar las limitaciones con el nombre CHECK.

```
CREATE TABLE Purchase
( PurchNo          CHAR(8),
  PurchDate        DATE CONSTRAINT PurchDateRequired NOT NULL,
  SuppNo           CHAR(8) CONSTRAINT SuppNo2Required NOT NULL,
  PurchPayMethod   CHAR(6) DEFAULT 'PO',
  PurchDelDate     DATE,
  CONSTRAINT PKPurchase PRIMARY KEY (PurchNo),
  CONSTRAINT SuppNoFK2 FOREIGN KEY (SuppNo) REFERENCES Supplier )
```

- La fecha de entrega de la compra es posterior a la fecha de compra o nula.
  - El método de pago de la compra no es nulo cuando la fecha de entrega de la compra no es nulo.
  - El método de pago es PO, CC, DC o nulo.
11. En este problema, deberá crear una vista, varias funciones y luego otorgar tipos de acceso específicos de la vista para las funciones.
    - Cree una vista de la tabla *Supplier* en la base de datos de captura de pedidos extendida que presentamos en la sección de problemas del capítulo 10. La vista debe incluir todas las columnas de la tabla *Supplier* para los proveedores de productos de impresión (la columna *Product.ProdName* que contiene la palabra “Printer”). Su vista se debe llamar “PrinterSupplierView”.
    - Defina tres funciones: *PrinterProductEmp*, *PrinterProductMgr* y *StoreMgr*.
    - Otorgue los siguientes privilegios de *PrinterSupplierView* a *PrinterProductEmp*: recuperación de todas las columnas excepto la del descuento a proveedores.
    - Otorgue los siguientes privilegios de *PrinterSupplierView* a *PrinterProductMgr*: recuperación y modificación de todas las columnas de *PrinterSupplierView*, excepto la del descuento a proveedores.
    - Otorgue los siguientes privilegios de *PrinterSupplierView* a *StoreMgr*: recuperación de todas las columnas, insertar, eliminar y modificar el descuento a proveedores.
  12. Identifique los privilegios más importantes en un DBMS empresarial para *data warehouses* y estadísticas de bases de datos. Los privilegios son específicos de los fabricantes, de modo que debe leer la documentación de un DBMS empresarial.
  13. Identifique y describa en forma breve las tablas de diccionario para estadísticas de bases de datos en un DBMS empresarial. Las tablas de diccionario son específicas de los fabricantes, de modo que debe leer la documentación de un DBMS empresarial.
  14. Escriba un breve resumen (una página) de los privilegios del DBA en un DBMS empresarial. Identifique las funciones definidas y/o las cuentas de usuario con privilegios de DBA, así como los privilegios otorgados a estas funciones.

## Referencias para ampliar su estudio

El libro de Jay Louise Weldon (1981), a pesar del tiempo, sigue siendo una obra clásica sobre la administración de bases de datos. Mullin (2002) ofrece una referencia completa más reciente sobre la administración de bases de datos. La sección “Administración de recursos de información” de la lista en línea de recursos web ofrece ligas a fuentes sobre administración de recursos de información y administración del conocimiento. Varios libros de SQL proporcionan detalles adicionales sobre las características de seguridad e integridad en SQL. Inmon (1986) y Martin (1982) escribieron descripciones detalladas de la planeación de sistemas de información. Castano *et al.* (1995) es una buena referencia para detalles adicionales sobre seguridad de bases de datos. Para más detalles acerca del proceso jerárquico analítico que mencionamos en la sección 14.3.2, consulte Saaty (1988) y Zahedi (1986). Su *et al.* (1987) describen la calificación lógica de las preferencias, un planteamiento alternativo para la selección de DBMS. El *Transaction Processing Council* ([www.tpc.org](http://www.tpc.org)) ofrece un valioso recurso sobre las evaluaciones comparativas específicas de cada dominio para los DBMS.

## Apéndice 14.A

### Resumen de la sintaxis SQL:2003

Este apéndice resume la sintaxis SQL:2003 para las sentencias CREATE/DROP ROLE, las sentencias GRANT/REVOKE, la sentencia CREATE DOMAIN y las sentencias CREATE ASSERTION, así como la cláusula de limitación CHECK de la sentencia CREATE TABLE. Las convenciones utilizadas en la sintaxis son idénticas a aquellas que se usan al final del capítulo 3.

### Sentencias CREATE y DROP ROLE

```
CREATE ROLE RoleName
    [ WITH ADMIN UserName{ CURRENT_USER | CURRENT_ROLE } ]
```

```
DROP ROLE RoleName
```

### Sentencias GRANT y REVOKE

```
-- GRANT statement for privileges
GRANT { <Privilege>* | ALL PRIVILEGES } ON ObjectName
    TO UserName* [ WITH GRANT OPTION ]
```

<Privilege>:

```
{ SELECT [ (ColumnName*) ] |
    DELETE |
    INSERT [ (ColumnName*) ] |
    REFERENCES [ (ColumnName*) ] |
    UPDATE [ (ColumnName*) ] |
    USAGE |
    TRIGGER |
    UNDER |
    EXECUTE }
```

```
-- GRANT statement for roles
GRANT RoleName*
    TO UserName* [ WITH ADMIN OPTION ]]

-- REVOKE statement for privileges
REVOKE [ GRANT OPTION FOR ] <Privilege>*
    ON ObjectName FROM UserName*
    [ GRANTED BY { CURRENT_USER | CURRENT_ROLE } ]
    { CASCADE | RESTRICT }

-- REVOKE statement for roles
REVOKE [ ADMIN OPTION FOR ] RoleName*
    FROM UserName*
    [ GRANTED BY { CURRENT_USER | CURRENT_ROLE } ]
    { CASCADE | RESTRICT }
```

## Sentencias CREATE DOMAIN y DROP DOMAIN

```
CREATE DOMAIN DomainName DataType
[ CHECK ( <Domain-Condition> ) ]

<Domain-Condition>:
{ VALUE <Comparison-Operator> Constant | 
  VALUE BETWEEN Constant AND Constant | 
  VALUE IN ( Constant* ) }

<Comparison-Operator>:
{ = | < | > | <= | >= | <> }
```

DROP DOMAIN DomainName { CASCADE | RESTRICT }

## Sentencias CREATE ASSERTION y DROP ASSERTION

```
CREATE ASSERTION AssertionName
    CHECK ( <Group-Condition> )

<Group-Condition>: -- initially defined in Chapter 4 and extended in Chapter 9

DROP ASSERTION AssertionName { CASCADE | RESTRICT }
```

## Cláusula de limitación CHECK en la sentencia CREATE TABLE

```

CREATE TABLE TableName
  ( <Column-Definition>* [ , <Table-Constraint>* ] )

<Column-Definition>: ColumnName DataType
    [ DEFAULT { DefaultValue | USER | NULL } ]
    [ <Embedded-Column-Constraint>+ ]

-- Check constraint can be used as an embedded column
-- constraint or as a table constraint.

<Embedded-Column-Constraint>:
{ [ CONSTRAINT ConstraintName ] NOT NULL |
  [ CONSTRAINT ConstraintName ] UNIQUE |
  [ CONSTRAINT ConstraintName ] <Check-Constraint> |
  [ CONSTRAINT ConstraintName ] PRIMARY KEY |
  [ CONSTRAINT ConstraintName ] FOREIGN KEY
    REFERENCES TableName [ ( ColumnName ) ]
    [ ON DELETE <Action-Specification> ]
    [ ON UPDATE <Action-Specification> ] }

<Table-Constraint>: [ CONSTRAINT ConstraintName ]
{ <Primary-Key-Constraint> |
  <Foreign-Key-Constraint> |
  <Uniqueness-Constraint> |
  <Check-Constraint> }

<Primary-Key-Constraint>: PRIMARY KEY (ColumnName* )

<Foreign-Key-Constraint>: FOREIGN KEY ( ColumnName* )
  REFERENCES TableName [ ( ColumnName* ) ]
  [ ON DELETE <Action-Specification> ]
  [ ON UPDATE <Action-Specification> ]

<Action-Specification>: { CASCADE | SET NULL | SET DEFAULT | RESTRICT }

<Uniqueness-Constraint>: UNIQUE (ColumnName* )

<Check-Constraint>: CHECK ( <Row-Condition> )

<Row-Condition>: -- defined in Chapter 4

```



# Capítulo 15

## Administración de transacciones

### Objetivos de aprendizaje

Este capítulo describe las características de la administración de transacciones para soportar el uso concurrente de una base de datos y la recuperación en caso de falla. Al terminar este capítulo, el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Explicar las propiedades de las transacciones ACID y los conceptos de recuperación y transparencia de concurrencia.
- Entender la función de los candados para evitar problemas de interferencia entre varios usuarios.
- Entender la función de las herramientas de recuperación para manejar las fallas en las bases de datos.
- Entender los aspectos del diseño de transacciones que afectan el desempeño.
- Describir la relación de la administración del flujo de trabajo con la administración de transacciones.

### Panorama general

El capítulo 14 presentó el contexto para administrar bases de datos y un panorama general de los distintos ambientes de procesamiento para bases de datos. Usted aprendió las responsabilidades de los especialistas en bases de datos, así como las herramientas y procesos que estos especialistas utilizan. El ambiente de bases de datos más común e importante es el procesamiento de transacciones, que ofrece soporte para las operaciones cotidianas de una organización. Este capítulo empieza con los detalles de la parte 7, describiendo la forma en que los DBMS soportan el procesamiento de transacciones.

Este capítulo presenta una amplia cobertura de la administración de transacciones. Antes de entender el soporte de los DBMS para el procesamiento de transacciones, usted necesita un conocimiento más detallado de los conceptos de las transacciones. Este capítulo describe las propiedades de las transacciones, enunciados SQL para definir las transacciones y las propiedades del procesamiento de transacciones. Después de aprender los conceptos de las transacciones, estará listo para estudiar el control de concurrencias y el manejo de recuperaciones, dos importantes servicios para apoyar el procesamiento de transacciones. En cuanto al control de concurrencias, este capítulo describe el objetivo, los problemas de interferencia y las herramientas de control de concurrencias. En relación con el manejo de recuperaciones, este capítulo describe los tipos de fallas, las herramientas y los procesos de recuperación.

Además de conocer los servicios de administración de transacciones que ofrece un DBMS, debe entender los aspectos del diseño de transacciones. Este capítulo describe importantes aspectos del diseño de transacciones e incluye puntos decisivos, límites de transacciones, niveles de aislamiento y aplicación de la limitación de integridad. Para ampliar sus antecedentes, debe entender cómo las transacciones de bases de datos se integran en el amplio contexto del trabajo colaborativo. La última sección describe la administración del flujo de trabajo y la compara con la administración de las transacciones en los DBMS.

## 15.1 Aspectos básicos de las transacciones de bases de datos

### **transacción**

unidad de trabajo que se debe procesar en forma confiable. Los DBMS ofrecen servicios de recuperación y control de concurrencias para procesar las transacciones de manera eficiente y confiable.

El procesamiento de transacciones comprende el lado operativo de las bases de datos. Mientras que la administración de operaciones describe la forma en que se producen los bienes físicos, la administración de transacciones describe la manera en que se controlan los bienes de información o transacciones. La administración de transacciones, al igual que la administración de bienes físicos, es muy importante para las organizaciones modernas. Organizaciones como bancos con cajeros automáticos, líneas aéreas con sistemas de reservaciones y universidades con registros en línea, no podrían funcionar sin un procesamiento de transacciones confiable y eficiente. En la actualidad, las organizaciones grandes realizan miles de transacciones por minuto. Con el crecimiento continuo del comercio electrónico, aumenta la importancia del procesamiento de transacciones.

En un lenguaje común, una transacción es una interacción entre dos o más partes para realizar un negocio, como la compra de un auto a una distribuidora. Las transacciones de bases de datos tienen un significado más preciso. Una transacción de base de datos comprende un grupo de operaciones que se deben procesar como una unidad de trabajo. Las transacciones se deben procesar de manera confiable, de modo que no haya ninguna pérdida de datos debido a usuarios múltiples o fallas en el sistema. Para ayudarle a entender con mayor precisión este significado, esta sección presenta ejemplos de transacciones y define sus propiedades.

### 15.1.1 Ejemplos de transacciones

Una transacción es un concepto definido por el usuario. Por ejemplo, hacer una reservación en una línea aérea comprende las reservaciones para la salida y la llegada. Para el usuario, la combinación de la salida y la llegada es una transacción, en lugar de la salida y la llegada por separado. La mayoría de los viajeros no quieren salir sin regresar. La implicación para los DBMS es que una transacción es un conjunto de operaciones de bases de datos definidas por el usuario. Una transacción puede comprender cualquier cantidad de lecturas y escrituras en una base de datos. Con el fin de ofrecer flexibilidad en las transacciones definidas por el usuario, los DBMS no pueden limitar las transacciones a un número específico de lecturas y escrituras en la base de datos.

Un sistema de información puede tener varias clases de transacciones diferentes. La tabla 15.1 ilustra las transacciones de un sistema de captura de pedidos. En cualquier momento los clientes podrían crear negocios con cada una de estas transacciones. Por ejemplo, muchos clientes pueden hacer pedidos mientras otros revisan el estado de sus pedidos. Como ejemplo adicional de las transacciones en un sistema de información, la tabla 15.2 ilustra las transacciones típicas en el sistema de nómina de una universidad. Algunas de estas transacciones son periódicas, mientras otras son eventos únicos.

#### *Enunciados SQL para definir las transacciones*

Para definir una transacción puede utilizar algunos enunciados SQL adicionales. La figura 15.1 ilustra enunciados SQL adicionales para definir el prototipo de una transacción en un

**TABLA 15.1**  
Transacciones típicas  
en un sistema de cap-  
tura de pedidos

Transacción	Descripción
Agregar pedido	El cliente hace un nuevo pedido
Actualizar pedido	El cliente cambia los detalles de un pedido existente
Revisar el estado	El cliente revisa el estado de un pedido
Pago	Se recibe el pago del cliente
Envío	El bien se envía al cliente

**TABLA 15.2**  
**Transacciones típicas**  
**en el sistema de nómina de una universidad**

Transacción	Descripción
Contratar un empleado	El empleado inicia sus servicios en la universidad
Pagar al empleado	Pago periódico realizado al empleado por sus servicios
Presentar registro de horas	Los empleados por hora presentan un registro de las horas trabajadas
Reasignación	Se asigna un nuevo puesto al empleado
Evaluación	Evaluación periódica de desempeño
Terminación	El empleado deja el trabajo en la universidad

**FIGURA 15.1**  
**Seudocódigo para una transacción en cajero automático**

```

INICIA TRANSACCIÓN
Muestra bienvenida
Obtener número de cuenta, NIP, tipo y cantidad
SELECT número de cuenta, tipo y saldo
If saldo es suficiente then
    UPDATE cuenta anotando el débito
    UPDATE cuenta anotando el crédito
    INSERT registro histórico
    Mostrar mensaje final y entrega de efectivo
Else
    Escribir mensaje de error
End If
On Error: ROLLBACK
COMMIT

```

cajero automático (ATM). Los enunciados START TRANSACTION y COMMIT definen los enunciados en una transacción.<sup>1</sup> Cualquier otro enunciado SQL entre ellos forma parte de la transacción. Por lo regular, una transacción comprende varios enunciados SELECT, INSERT, UPDATE y DELETE. En la figura 15.1, una transacción real tendría enunciados SQL válidos para las filas que empiezan con SELECT, UPDATE e INSERT. Los enunciados válidos en el lenguaje de programación deben sustituirse por líneas con comillas, como “Display greeting”.

Además de los enunciados START TRANSACTION y COMMIT, es posible usar el enunciado ROLLBACK. ROLLBACK es como el comando deshacer en un procesador de texto, en el sentido de que se eliminan los efectos de las acciones del usuario. A diferencia del comando deshacer, ROLLBACK se aplica a una secuencia de acciones y no sólo a una acción. Por lo tanto, un enunciado ROLLBACK provoca la eliminación de todos los efectos de una transacción. La base de datos se restaura al estado en que se encontraba antes de ejecutar la transacción.

Un enunciado ROLLBACK puede usarse en varios contextos. Una situación es permitir que un usuario cancele una transacción. Una segunda es responder a los errores. En esta situación, el enunciado ROLLBACK puede usarse como parte de los enunciados de manejo de excepciones, como la línea “On Error” de la figura 15.1. Los enunciados de manejo de excepciones forman parte de lenguajes de programación como Java y Visual Basic. El manejo de excepciones permite que errores no anticipados, como los de comunicación, se procesen por separado a partir de la lógica normal de la transacción.

Como aprenderá más adelante en este capítulo, las transacciones deben estar diseñadas para durar poco. Con el fin de acortar la duración de una transacción en un cajero automático, la interacción del usuario debe colocarse fuera de la transacción siempre que sea posible. En una transacción en cajero automático, el enunciado START TRANSACTION se puede colocar después de las primeras tres filas para eliminar la interacción con el usuario. Las transacciones muy largas pueden dar lugar a un tiempo de espera prolongado entre los usuarios concurrentes de una base de datos. Sin embargo, los enunciados UPDATE e INSERT deben permanecer en la misma transacción porque forman parte de la misma unidad de trabajo.

<sup>1</sup> SQL:2003 especifica los enunciados START TRANSACTION y COMMIT. Algunos DBMS utilizan la palabra clave BEGIN en lugar de START. Algunos DBMS, como Oracle, no usan un enunciado para iniciar una transacción de manera explícita. Una nueva transacción empieza con el siguiente enunciado SQL seguido del enunciado COMMIT.

**FIGURA 15.2**  
Pseudocódigo para una transacción de reservación en una línea aérea

```

INICIA TRANSACCIÓN
    Muestra bienvenida
    Obtener preferencias de reservación
    SELECT registro de vuelos de salida y regreso
    If la reservación es aceptable then
        UPDATE asientos disponibles del registro de salida de vuelos
        UPDATE asientos disponibles del registro de llegada de vuelos
        INSERT registro de reservación
        Imprimir boleto si se solicita
    End If
    On Error: ROLLBACK
COMMIT

```

**FIGURA 15.3**  
Pseudocódigo para una transacción de pedido de productos

```

INICIA TRANSACCIÓN
    Muestra bienvenida
    Obtener solicitud de orden
    SELECT registro de producto
    If el producto está disponible then
        UPDATE QOH del registro del producto
        INSERT registro de orden
        Enviar mensaje al departamento de envíos
    End If
    On Error: ROLLBACK
COMMIT

```

#### *Otros ejemplos de transacciones*

Las figuras 15.2 y 15.3 ilustran las transacciones para una reservación en una línea área y un pedido de productos. En ambos ejemplos, la transacción consiste en más de una acción en la base de datos (leer o escribir).

#### 15.1.2 Propiedades de la transacción

Los DBMS se aseguran de que las transacciones tengan ciertas propiedades. Como estudiaremos a continuación, las más importantes y conocidas son las propiedades ACID (atómica, consistente, aislada (isolated) y durable).

- **Atómica** significa que una transacción no se puede subdividir. Ya sea que se realice todo el trabajo en la transacción o que no se haga nada. Por ejemplo, la transacción en un cajero electrónico no representará débito en una cuenta sin también acreditar una cuenta correspondiente. La propiedad atómica implica que los cambios parciales que realiza una transacción deben deshacerse si ésta se cancela.
- **Consistente** significa que si las limitaciones aplicables son ciertas antes de empezar la transacción, éstas también lo serán al terminarla. Por ejemplo, si se calcula el saldo de la cuenta del usuario antes de la transacción, también se calcula después de ésta. De lo contrario, la transacción se rechaza y ningún cambio tiene lugar.
- **Aislada** significa que las transacciones no interfieren entre sí, excepto en formas permitidas. Una transacción nunca debe sobrescribir los cambios realizados por otra. Además, una transacción no debe interferir en otros aspectos, como no ver los cambios temporales realizados por otras transacciones. Por ejemplo, su pareja no sabrá que usted retiró dinero sino hasta terminar la transacción en el cajero electrónico.
- **Durable** significa que cualquier cambio que resulte de una transacción es permanente. Ninguna falla va a borrar ningún cambio después de terminar la transacción. Por ejemplo, si la computadora de un banco experimenta una falla cinco minutos después de terminar una transacción, sus resultados siguen grabados en la base de datos del banco.

Para asegurarse de que las transacciones tienen las propiedades ACID, los DBMS ofrecen ciertos servicios que son transparentes para los desarrolladores de bases de datos (programadores y analistas). En un lenguaje común, transparencia significa que es posible ver a través de un objeto, dejando invisibles los detalles internos. Para los DBMS, transparencia significa que los detalles internos de los servicios de las transacciones son invisibles. La transparencia es muy importante porque los servicios que garantizan las transacciones ACID son difíciles de implementar. Al prestar estos servicios, los DBMS mejoran la productividad de los programadores y analistas de bases de datos.

Los DBMS ofrecen dos servicios para garantizar que las transacciones tienen las propiedades ACID: transparencia de recuperación y transparencia de concurrencia. La recuperación comprende acciones para manejar las fallas, como errores en la comunicación y colapsos del software. La concurrencia comprende acciones para controlar la interferencia entre varios usuarios simultáneos de la base de datos. El análisis siguiente proporciona los detalles acerca de la transparencia de recuperación y concurrencia.

- La **transparencia de recuperación** significa que el DBMS restaura automáticamente una base de datos a un estado consistente después de una falla. Por ejemplo, si ocurre una falla de comunicación durante una transacción en un cajero automático, los efectos de la transacción quedan borrados de la base de datos. Por otra parte, si el DBMS se colapsa tres segundos después de que termina una transacción en un cajero automático, los detalles de la transacción son permanentes.
- La **transparencia de concurrencia** significa que los usuarios perciben la base de datos como un sistema para un solo usuario, aunque haya varios usuarios simultáneos. Por ejemplo, aun cuando muchos usuarios traten de reservar lugares en el mismo vuelo mediante una transacción de reserva, el DBMS se asegura de que cada usuario no sobrescriba el trabajo de los demás.

Aun cuando los detalles internos de la concurrencia y la recuperación no sean visibles para un usuario, estos servicios no son gratuitos. El control de la recuperación y la concurrencia comprende gastos generales que quizás requieren de recursos adicionales y de una supervisión estrecha para llegar a un nivel de desempeño aceptable. El DBA debe estar consciente de las implicaciones de estos servicios en los recursos. Es probable mejorar el desempeño con una mayor cantidad de recursos de equipo de cómputo, como memoria, espacio en disco y procesamiento paralelo. Es necesario monitorear el desempeño para garantizar su adecuado funcionamiento. El DBA debe vigilar los indicadores clave del desempeño y cambiar los valores de los parámetros para reducir los problemas de desempeño.

Además de los recursos y el monitoreo, la selección de un DBMS puede ser crucial para lograr un desempeño aceptable en el procesamiento de las transacciones. A menudo, el precio de compra de un DBMS depende del número de transacciones concurrentes para las que ofrece soporte. Los DBMS que soportan gran cantidad de usuarios concurrentes pueden ser costosos. El diseño de transacciones es otra razón para entender los detalles del control de la concurrencia y la recuperación. Aun cuando seleccione un DBMS eficiente y dedique los recursos adecuados al procesamiento de transacciones, un mal diseño de las transacciones puede dar lugar a problemas de desempeño. Para lograr un diseño de transacciones satisfactorio, es preciso conocer los detalles del control de la concurrencia y la recuperación, así como entender los principios de diseño de las transacciones, como estudiaremos en las secciones siguientes.

## 15.2 Control de la concurrencia

---

La mayoría de las organizaciones no podrían funcionar sin bases de datos para usuarios múltiples. Por ejemplo, las bases de datos de líneas aéreas, tiendas detallistas, bancos y servicios de ayuda pueden tener miles de usuarios que tratan de hacer negociaciones al mismo tiempo. En estas bases de datos varios usuarios tienen acceso concurrente, es decir, al mismo tiempo. Si el acceso estuviera limitado a un usuario a la vez, se realizaría una parte mínima del trabajo y la mayoría de los usuarios se llevarían su negocio a otra parte. Sin embargo, los usuarios concurrentes no

se pueden interferir entre ellos. Esta sección define el objetivo, los problemas y las herramientas para el control de la concurrencia.

### 15.2.1 Objetivo del control de concurrencia

**caudal de procesamiento de transacciones**

número de transacciones procesadas por intervalo. Es una importante medida del desempeño del procesamiento de las transacciones.

El objetivo del control de la concurrencia es maximizar el caudal de procesamiento de transacciones y, al mismo tiempo, evitar la interferencia entre varios usuarios. El caudal de procesamiento de transacciones, el número de transacciones procesadas por unidad de tiempo, es una importante medida del trabajo que realiza un DBMS. Por lo regular, el caudal de procesamiento se reporta en transacciones por minuto. En un ambiente de alto volumen, como el comercio electrónico, los DBMS quizá necesiten procesar miles de transacciones por minuto. El Transaction Processing Council ([www.tpc.org](http://www.tpc.org)) reporta los resultados para la evaluación TPC-C comparativa (captura de pedidos) de más de un millón de transacciones por segundo.

Desde la perspectiva del usuario, el caudal de procesamiento de transacciones está relacionado con el tiempo de respuesta. Un caudal de procesamiento más alto significa tiempos de respuesta más largos. Por lo general, los usuarios no están dispuestos a esperar más de unos cuantos segundos para la realización de una transacción.

Si no hay ninguna interferencia, el resultado de ejecutar las transacciones concurrentes es el mismo que el de ejecutar las mismas transacciones en orden secuencial. La ejecución secuencial significa que una transacción finaliza antes de que otra se ejecute, garantizando así que no habrá ninguna interferencia. El hecho de ejecutar las transacciones en forma secuencial daría como resultado un caudal de procesamiento bajo y tiempos de espera prolongados. Por lo tanto, los DBMS permiten que las transacciones se ejecuten al mismo tiempo y garantizan que los resultados serán los mismos que si se llevaran a cabo en forma secuencial.

Las transacciones que se ejecutan de manera simultánea no pueden interferir entre sí, a menos que manipulen datos comunes. La mayor parte de las transacciones concurrentes manipulan sólo pequeñas cantidades de datos comunes. Por ejemplo, en una reservación en una línea aérea, dos usuarios pueden capturar al mismo tiempo nuevos registros de reservaciones porque éstos son únicos para cada cliente. Sin embargo, podría ocurrir una interferencia en la columna de asientos libres de la tabla de un vuelo. Para los vuelos con mayor demanda, muchos usuarios quieren reducir el valor de la columna de asientos libres. Es muy importante que el DBMS controle la actualización concurrente de esta columna en los registros de los vuelos con mayor demanda.

Un punto decisivo son los datos comunes que varios usuarios tratan de cambiar al mismo tiempo. En esencia, un punto decisivo representa un recurso escaso por el que los usuarios deben esperar. Los puntos decisivos típicos son los asientos libres en los vuelos de mayor demanda, la cantidad disponible de artículos populares en un inventario y los lugares disponibles en los cursos ofrecidos. En un mundo ideal, los DBMS sólo registrarían los puntos decisivos, pero, por desgracia, puede ser difícil conocerlos por anticipado, de modo que los DBMS registran el acceso a cualquier parte de una base de datos.

La interferencia en los puntos decisivos puede dar lugar a datos perdidos y a la toma de decisiones equivocadas. Las secciones siguientes describen los problemas de interferencia y las herramientas para prevenirlos.

### 15.2.2 Problemas de interferencia

Hay tres problemas que se pueden presentar debido al acceso simultáneo a una base de datos: (1) actualización perdida, (2) dependencia sin realizaciones y (3) recuperación inconsistente. Esta sección define cada problema y presenta ejemplos de su ocurrencia.

#### *Actualización perdida*

**actualización perdida**

un problema de control de concurrencia en el que la actualización de un usuario sobrescribe la de otro.

Una actualización perdida es el problema de interferencia más serio porque los cambios a una base de datos se pierden sin que nadie se dé cuenta. En una actualización perdida, la actualización de un usuario sobrescribe la de otro, como ilustra la cronología de la figura 15.4. La cronología muestra dos transacciones concurrentes que tratan de actualizar el campo de asientos libres (*SR*) en el registro del mismo vuelo. Supongamos que el valor de *SR* es 10 antes de que empiecen las transacciones. Después del tiempo  $T_2$ , ambas transacciones guardan el valor de 10 para *SR* en los búferes locales, como resultado de las operaciones leídas. Después del tiempo  $T_4$ , ambas

**FIGURA 15.4**  
Ejemplo de un problema de actualización perdida

Transacción A	Tiempo	Transacción B
Leer SR (10)	$T_1$	
	$T_2$	Leer SR (10)
Si $SR > 0$ entonces $SR = SR - 1$	$T_3$	
	$T_4$	Si $SR > 0$ entonces $SR = SR - 1$
Escribir SR (9)	$T_5$	
	$T_6$	Escribir SR (9)

**FIGURA 15.5**  
Ejemplo de un problema de lectura sucia

Transacción A	Tiempo	Transacción B
Leer SR (10)	$T_1$	
$SR = SR - 1$	$T_2$	
Escribir SR (9)	$T_3$	
	$T_4$	Leer SR (9)
Rollback	$T_5$	

transacciones realizan cambios a su copia local de  $SR$ . Sin embargo, cada transacción cambia el valor a 9, sin darse cuenta de la actividad de la otra transacción. Después del tiempo  $T_6$ , el valor de  $SR$  en la base de datos es 9. Pero el valor después de terminar ambas transacciones debe ser 8 y no 9. Se perdió uno de los cambios.

Algunos estudiantes se confunden con el problema de la actualización perdida por las acciones realizadas en copias locales de los datos. Los cálculos en los tiempos  $T_3$  y  $T_4$  se realizan en los búferes de la memoria específicos para cada transacción. Aun cuando la transacción A cambió el valor de  $SR$ , la transacción B realiza el cálculo con una copia local de  $SR$  que tiene un valor de 10. La transacción B ni se entera de la escritura de la operación que realiza la transacción A, a menos que la transacción B vuelva a leer el valor.

Una actualización perdida comprende dos o más transacciones que tratan de cambiar (escribir en) la misma parte de la base de datos. Como verá en los dos problemas siguientes, dos transacciones también pueden entrar en conflicto si sólo una de ellas cambia la base de datos.

#### *Dependencia sin realización*

#### **dependencia sin realización**

problema de control de concurrencia en el que una transacción lee los datos que escribe otra antes de que la otra transacción se realice. Si la segunda transacción se cancela, la primera transacción puede depender de datos que ya no existen.

Una dependencia sin realización sucede cuando una transacción lee los datos que escribe otra antes de que la otra transacción se realice. Una dependencia sin realización también se conoce como lectura sucia porque la provoca una transacción que lee datos sucios (sin realizar). En la figura 15.5, la transacción A lee el campo  $SR$ , cambia su copia local del campo  $SR$  y escribe el nuevo valor en la base de datos. Luego, la transacción B lee el valor cambiado. Sin embargo, antes de la transacción A, se detecta un error y la transacción A emite una restauración al estado original. Esta restauración podría haberse emitido como resultado de la cancelación de la transacción por parte del usuario o de una falla. Ahora, el valor que utiliza la transacción B es un valor fantasma. El verdadero valor  $SR$  es 10 porque el cambio de A no es permanente. La transacción B puede utilizar su valor (9) para tomar una decisión errónea. Por ejemplo, si el valor de  $SR$  era 1 antes de iniciar la transacción A, es probable que a la transacción B se le niegue una reservación.

Como los datos no son permanentes hasta que se realiza una transacción, quizás ocurra un conflicto aun cuando sólo una transacción escriba en la base de datos. Una dependencia sin realización comprende una transacción que escribe y otra que lee en la misma parte de la base de datos. Sin embargo, una dependencia sin realización no puede ocasionar un problema a menos que ocurra una restauración al estado original. El tercer problema también representa conflictos cuando una sola transacción escribe en la base de datos.

#### *Problemas que comprenden recuperaciones inconsistentes*

El último problema comprende situaciones en las que la interferencia provoca inconsistencias entre varias recuperaciones de un subconjunto de datos. Todos los problemas de recuperaciones inconsistentes comprenden una transacción que lee y una segunda transacción que cambia la

**FIGURA 15.6**  
**Ejemplo de un problema de resumen incorrecto**

Transacción A	Tiempo	Transacción B
Leer $SR_1$ (10)	$T_1$	
$SR_1 = SR_1 - 1$	$T_2$	
Escribir $SR_1$ (9)	$T_3$	
	$T_4$	Leer $SR_1$ (9)
	$T_5$	Sumar = Suma + $SR_1$
	$T_6$	Leer $SR_2$ (5)
	$T_7$	Sumar = Suma + $SR_2$
Leer $SR_2$ (5)	$T_8$	
$SR_2 = SR_2 - 1$	$T_9$	
Escribir $SR_2$ (4)	$T_{10}$	

**resumen incorrecto** problema de control de concurrencia en el que una transacción lee diversos valores, pero otra actualiza algunos de los valores mientras la primera todavía se está ejecutando.

misma parte de la base de datos. El problema de resumen incorrecto es el más importante entre los que comprenden recuperaciones inconsistentes. Un resumen incorrecto ocurre cuando una transacción, mientras calcula la función de resumen, lee algunos valores antes de que otra transacción los cambie, pero lee otros valores después de que otra transacción los cambió.<sup>2</sup> En la figura 15.6, la transacción B lee  $SR_1$  después de que la transacción A cambió el valor, pero lee  $SR_2$  antes de que la transacción A lo cambie. Para que haya consistencia, la transacción B debe utilizar todos los valores antes o después de que otras transacciones los cambien.

Un segundo problema que comprende recuperaciones inconsistentes se conoce como problema de lectura fantasma. Un problema de lectura fantasma se presenta cuando una transacción realiza una consulta con condiciones de registro. Luego, otra transacción inserta o modifica los datos que la consulta recuperaría. Por último, la transacción original ejecuta la misma consulta otra vez. La segunda ejecución de la consulta recupera distintos registros que los que recuperó la primera. Los registros nuevos y cambiados son fantasmas porque no existen en el resultado de la ejecución de la primera consulta.

Un tercer problema que comprende recuperaciones inconsistentes se conoce como problema de lectura no repetible. Un problema de lectura no repetible ocurre cuando una transacción lee el mismo valor más de una vez. Entre una y otra lectura de datos, otra transacción los modifica. La segunda recuperación contiene un valor diferente que la primera por el cambio que realiza otra transacción.

Los problemas de lectura no repetible y fantasma son ligeramente diferentes. Un problema de lectura no repetible ocurre si otro usuario cambia el valor de una columna de una fila de consulta, de modo que la consulta regresa un valor diferente en la siguiente ejecución. Un problema de lectura fantasma ocurriría al insertar una nueva fila que coincide con una condición en una consulta, de modo que esta última recupera una fila adicional en la siguiente ejecución. La diferencia clave es el requisito de la condición de la fila para el problema de lectura fantasma.

### 15.2.3 Herramientas de control de concurrencia

Esta sección describe dos herramientas, candados y el protocolo de candado de dos etapas, que utilizan la mayoría de los DBMS para evitar los tres problemas de interferencia que estudiamos en la sección anterior. Además de las dos herramientas, se presenta el problema del punto muerto (deadlock) como resultado del uso de candados. Esta sección termina con el breve estudio de planteamientos optimistas de control de concurrencia que no utilizan candados.

#### Candados

Los candados ofrecen una forma de evitar que otros usuarios tengan acceso a un elemento de una base de datos que está en uso. Un elemento de una base de datos puede ser una fila, un bloque, un subconjunto de filas o incluso una tabla completa. Antes de tener acceso al elemento de la base de datos, es necesario obtener un candado. Otros usuarios deben esperar si tratan de obtener un candado en el mismo elemento. La tabla 15.3 muestra conflictos para ambos tipos

<sup>2</sup> Un resumen incorrecto se conoce también como análisis inconsistente.

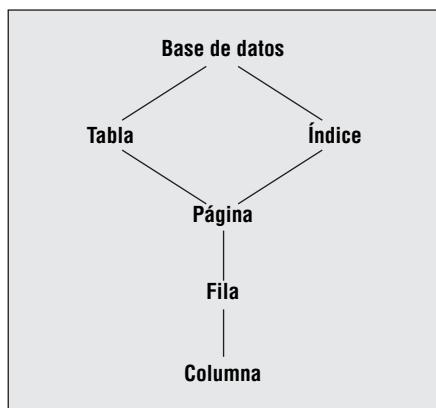
**TABLA 15.3**  
Conflictos de candados

Consultas del usuario 2		
Usuario 1 Espera	Candado S	Candado X
Candado S	Candado otorgado	Usuario 2 espera
Candado X	Usuario 2 espera	Usuario 2 espera

**TABLA 15.4**  
Campos en un registro de candados

Nombre del campo	Descripción
Identificador de transacciones	Identificador único para una transacción
Identificador de registro	Identifica el registro que se protegerá con candado
Tipo	Indica el uso del registro con candado
Cuenta	Número de usuarios que tienen este tipo de candados

**FIGURA 15.7**  
Niveles típicos de granularidad de los candados



### candado

herramienta fundamental del control de concurrencia. Un candado en un elemento de una base de datos evita que se realicen otras transacciones que ponen en conflicto las acciones en el mismo elemento.

### granularidad de candados

el tamaño del elemento de una base de datos con candados. La granularidad del candado es un intercambio entre el tiempo de espera (cantidad de concurrencia permitida) y el trabajo adicional (número de candados que tiene).

de candados. Es preciso obtener un candado compartido (S) antes de leer un elemento de una base de datos, mientras que necesitamos un candado exclusivo (X) antes de escribir en él. Como muestra la tabla 15.3, cualquier cantidad de usuarios puede tener un candado compartido en el mismo elemento. Sin embargo, sólo un usuario puede tener un candado exclusivo.

El administrador del control de concurrencia es la parte del DBMS responsable de administrar los candados. Este administrador conserva una tabla oculta para registrar los candados que tienen diversas transacciones.<sup>3</sup> Un registro de candados contiene un identificador de transacciones, un identificador de registros, un tipo y una cuenta, como se explica en la tabla 15.4. Como se dijo antes, en el esquema más simple, el tipo es compartido o exclusivo. La mayor parte de los DBMS tienen otros tipos de candados para mejorar la eficiencia y permitir mayor acceso concurrente. El administrador del control de concurrencias realiza dos operaciones en los registros de candados. El operador de candado agrega una fila en la tabla de candados, mientras que el operador para abrir candados u operador de liberación elimina una fila de la tabla de candados.

### *Granularidad de candados*

La granularidad de candados representa una complicación en relación con los candados. La granularidad se refiere al tamaño del elemento de la base de datos que tiene candado. La mayor parte de los DBMS manejan candados de distintas granularidades, como ilustra la figura 15.7. El candado más extenso abarca toda la base de datos. Si toda la base tiene un candado exclusivo, ningún otro usuario tiene acceso a la base de datos hasta quitarlo. Por otro lado, el candado más pequeño abarca sólo una columna individual. También puede haber candados en partes de la base de datos que no ven los usuarios en general. Por ejemplo, es posible tener candados en los índices y en las páginas (registros físicos).

<sup>3</sup> La tabla de candados permanece oculta a todos los usuarios, excepto para el administrador de control de concurrencia. En circunstancias especiales, el administrador de bases de datos tiene acceso a la tabla de candados.

La granularidad de los candados es un intercambio entre el trabajo adicional y la espera. El hecho de tener candados más ligeros reduce la espera entre los usuarios, pero aumenta el trabajo adicional porque es preciso manejar más candados. El hecho de tener candados más poderosos reduce su cantidad, pero incrementa el tiempo de espera. En algunos DBMS, el administrador de control de concurrencia trata de detectar el patrón de uso y promueve los candados si son necesarios. Por ejemplo, en un principio, el administrador de control de concurrencia puede otorgar candados de registro a una transacción, anticipando que sólo se van a cerrar unos cuantos registros. Si la transacción sigue solicitando candados, el componente de control de concurrencia puede promover que las haya en un subconjunto de filas o en toda la tabla.

### candado de intención

candado en un elemento extenso de una base de datos (como una tabla) que indique la intención de cerrar elementos más pequeños contenidos en el elemento más grande. Los candados de intención aligeran el bloqueo al poner candados más fuertes en los elementos y permiten la detección eficiente de los conflictos entre los candados en elementos de diversa granularidad.

### punto muerto

un problema de espera mutua que puede ocurrir al utilizar candados. Si el punto muerto no se corrige, las transacciones involucradas esperarán para siempre. Un DBMS puede controlar los puntos muertos mediante la detección o mediante una política de tiempo fuera.

Los candados de intención se utilizan por lo regular para aligerar el bloqueo en forma compartida o exclusiva causado por los candados más fuertes en los elementos. Los candados de intención soportan más concurrencia en los elementos pesados que los candados compartidos o exclusivos. Además, permiten la detección eficiente de los conflictos entre candados en elementos de diversa granularidad. Para soportar la lectura y la escritura en niveles de granularidad más bajos, se utilizan tres tipos de candados de intención: (1) candado de intención compartido, cuando se quieren leer elementos de nivel más bajo; (2) candado de intención exclusivo, cuando se van a escribir elementos de nivel más bajo, y (3) candados de intención compartidos y exclusivos, cuando la idea es leer y escribir elementos de nivel más bajo. Por ejemplo, una transacción debe solicitar un candado de intención compartido en una tabla en la que quiera leer las filas. El candado de intención compartido en la tabla sólo entra en conflicto con los candados exclusivos en la tabla o en elementos de menor granularidad.

### Puntos muertos

El uso de candados para evitar problemas de interferencia puede dar lugar a puntos muertos. Un punto muerto es un problema de espera mutua. Una transacción tiene un recurso que necesita una segunda transacción, y esta última ocupa un recurso que la primera necesita. La figura 15.8 ilustra un punto muerto entre dos transacciones que tratan de reservar asientos en un vuelo que comprende más de un ruta. La transacción A adquiere un candado exclusivo en la primera ruta (digamos de Denver a Chicago), seguida por la transacción B que adquiere un candado en la segunda ruta (digamos de Chicago a Nueva York). La transacción A trata de poner un candado en la segunda ruta, pero está bloqueada porque la transacción B tiene un candado exclusivo. De modo similar, la transacción B debe esperar con el fin de obtener un candado para la primera ruta. Los puntos muertos pueden comprender más de dos transacciones, pero el patrón es más complejo.

La mayoría de los DBMS empresariales realizan una detección de puntos muertos con el fin de controlarlos. Este tipo de candados puede detectarse buscando patrones cílicos de espera mutua. En la práctica, casi todos los puntos muertos comprenden dos o tres transacciones. Como la detección de puntos muertos puede consumir mucho tiempo de procesamiento, sólo se realiza a intervalos periódicos o se dispara cuando hay transacciones en espera. Por ejemplo, la detección de puntos muertos para la figura 15.8 puede llevarse a cabo cuando la transacción B se ve obligada a esperar. Al detectar un punto muerto, por lo regular la transacción más reciente tiene que volver a empezar (caso de B en la figura 15.8).

La mayor parte de los DBMS de escritorio emplean una política más sencilla de tiempo fuera para controlar los puntos muertos. En una política de tiempo fuera, el administrador de control de concurrencia cancela (con un enunciado ROLLBACK) cualquier transacción que haya estado en espera durante más de un tiempo específico. Debemos hacer notar que una política de tiempo fuera puede cancelar transacciones que no tienen puntos muertos. El intervalo de tiempo fuera debe ser suficientemente largo para que pocas transacciones sin puntos muertos tengan que esperar mucho.

**FIGURA 15.8**  
Ejemplo de un problema de puntos muertos

Transacción A	Tiempo	Transacción B
XLock SR <sub>1</sub>	T <sub>1</sub>	
	T <sub>2</sub>	XLock SR <sub>2</sub>
XLock SR <sub>2</sub> (espera)	T <sub>3</sub>	
	T <sub>4</sub>	XLock SR <sub>1</sub> (espera)

### Protocolo de candado en dos etapas

#### definición de 2PL

1. Antes de leer o escribir en un elemento de datos, la transacción debe adquirir el candado aplicable para el elemento de datos.
2. Espere si hay un candado problemático en el elemento de datos.
3. Después de abrir un candado, la transacción no adquiere ningún nuevo candado.

Para asegurarse de que no se presenten problemas de actualizaciones perdidas, el administrador de control de concurrencia pide que todas las transacciones sigan el protocolo de candados en dos etapas (2PL). *Protocolo* es un término elegante para referirnos a una regla de comportamiento de grupo. Un protocolo pide a todos los miembros de un grupo que se comporten de una manera específica. Para la comunicación humana, Las Reglas de Orden de Robert piden a todos los participantes en una reunión que sigan ciertas reglas. Para la comunicación de datos, los protocolos se aseguran de que los mensajes tengan un formato común que tanto el remitente como el destinatario puedan reconocer. Para el control de concurrencia, todas las transacciones deben seguir el protocolo 2PL para garantizar que no ocurran problemas de concurrencia. Los candados en dos etapas tienen tres condiciones, como menciona la nota al margen en esta página.

Las dos primeras condiciones se derivan del uso de los candados, como explicamos antes. La tercera es más sutil. Si se adquieren nuevos candados después de abrir las primeras, dos transacciones pueden obtenerlas y abrirlas siguiendo un patrón en el que ocurre un problema de control de concurrencia.

Por lo general, la tercera condición se simplifica, de modo que se conserven los candados exclusivos hasta el final de la transacción. En el momento de realizarla, se liberan los candados de la transacción. La figura 15.9 ilustra en forma gráfica el protocolo 2PL con la tercera condición simplificada. Al principio de la transacción (BOT; *beginning of a transaction*), ésta no tiene candados. Se inicia una etapa de crecimiento en la que la transacción adquiere candados, pero nunca libera ninguno. Al final de la transacción (EOT; *end of a transaction*), ocurre la etapa de reducción, en la que todos los candados se liberan al mismo tiempo. El hecho de simplificar la definición del 2PL facilita la implementación del protocolo, así como el difícil problema de predecir el momento en que una transacción libera las cerraduras.

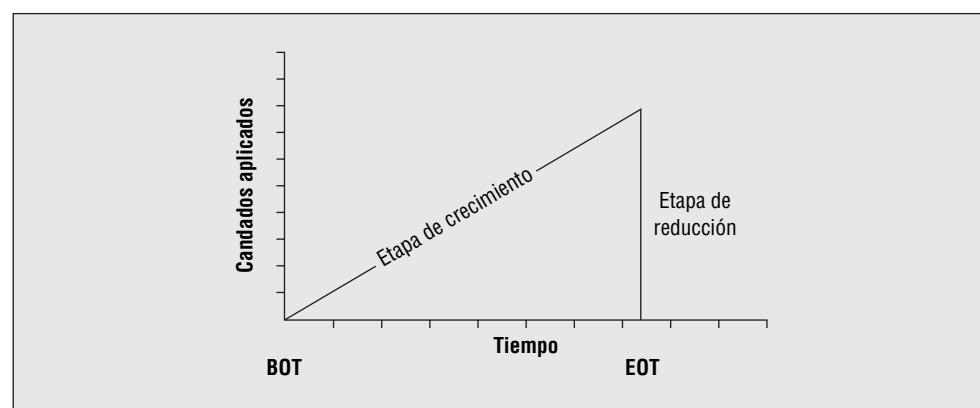
Para dar flexibilidad entre los problemas de control de concurrencia permitidos y potenciales que están en espera, la mayor parte de los DBMS relajan el protocolo 2PL para permitir que algunos candados se liberen antes del final de una transacción. La sección 15.4.2 presenta el concepto de los niveles de aislamiento para determinar el nivel de interferencia tolerado.

### Planteamientos optimistas

El uso de candados y el 2PL es un planteamiento pesimista para el control de concurrencias. El uso de candados supone que todas las transacciones entran en conflicto. Si la contención es sólo para unos cuantos puntos, es probable que los candados representen un trabajo adicional excesivo.

Los planteamientos de control de concurrencia optimistas suponen que los conflictos son raros. Si es así, es más eficiente revisar la presencia de conflictos que utilizar candados que obliguen a esperar. En los planteamientos optimistas, las transacciones pueden tener acceso a la base de datos sin adquirir candados. En vez de ello, el administrador de control de concurrencia revisa si ocurrió algún conflicto. La revisión se puede llevar a cabo antes de realizar la transacción o después de cada lectura y escritura. Revisando el tiempo relativo de las lecturas y escrituras, el administrador de control de concurrencia puede determinar si ha ocurrido algún conflicto. En

**FIGURA 15.9**  
Etapas de crecimiento y reducción del 2PL



caso de que así sea, el administrador indica que todo debe volver a su estado inicial y empezar la transacción que provocó el conflicto.

A pesar del atractivo de los planteamientos optimistas, la mayor parte de las organizaciones utilizan el 2PL aun cuando algunos DBMS empresariales ofrecen soporte para los planteamientos optimistas. El desempeño de estos planteamientos depende de la frecuencia de los conflictos. Si aumentan los conflictos, el desempeño baja. Aun cuando los conflictos sean raros, los planteamientos optimistas pueden tener mayor variabilidad porque el castigo por los conflictos es mayor en ellos. Los planteamientos pesimistas solucionan los conflictos mediante la espera, mientras que los optimistas los resuelven volviendo al estado inicial y empezando otra vez. El hecho de volver a empezar una transacción puede demorarla más que esperar la liberación de un recurso.

## 15.3 Administración de recuperaciones

---

La administración de recuperaciones es un servicio que restaura la base de datos a un estado consistente después de una falla. Esta sección describe los tipos de fallas que hay que prevenir, las herramientas para el manejo de recuperaciones y los procesos de recuperación que utilizan las herramientas.

### 15.3.1 Dispositivos de almacenamiento de datos y tipos de fallas

Desde la perspectiva de las fallas en las bases de datos, la volatilidad es una característica importante de los dispositivos para almacenamiento de datos. La memoria principal es volátil porque pierde su estado en caso de que no haya corriente eléctrica. En contraste, un disco duro es no volátil porque conserva su estado aun cuando no haya corriente eléctrica. Esta distinción es importante porque los DBMS no pueden depender de la memoria volátil para recuperar los datos después de las fallas. Ni siquiera los dispositivos no volátiles son totalmente confiables. Por ejemplo, ciertas fallas hacen que el contenido de un disco duro quede ilegible. Para lograr una alta confiabilidad, los DBMS pueden copiar los datos en diversos tipos de medios de almacenamiento no volátiles, como disco duro, cinta magnética y disco óptico. El hecho de usar una combinación de dispositivos no volátiles aumenta la confiabilidad porque, por lo regular, cada tipo de dispositivo tiene un índice de fallas independiente.

Algunas fallas afectan sólo la memoria principal, mientras que otras afectan la memoria volátil y la no volátil. La tabla 15.5 muestra cuatro tipos de fallas con sus efectos y frecuencia. Los primeros dos tipos de fallas afectan la memoria de una transacción en ejecución. El código de la transacción puede revisar que no haya condiciones de error, como un número de cuenta inválido o la cancelación de la transacción por parte del usuario. Por lo regular, una falla detectada por un programa da lugar al aborto de la transacción con un mensaje específico para el usuario. El enunciado SQL ROLLBACK puede cancelar la transacción si ocurre una condición anormal. Recuerde que el enunciado ROLLBACK hace que todos los cambios realizados por la transacción se eliminen de la base de datos. Las fallas que detecta el programa son, por lo general, las más comunes y menos dañinas.

La terminación anormal tiene un efecto similar para una falla detectada por un programa, mas por una causa diferente. La transacción se cancela, pero a menudo el usuario no puede entender el mensaje de error. La terminación anormal puede ser provocada por eventos como tiempo fuera en una transacción, fallas en la línea de comunicación o un error de programación (por ejemplo, dividir entre cero). El enunciado ON ERROR en la figura 15.1 detecta la terminación anormal. Un enunciado ROLLBACK elimina cualquier efecto de la transacción terminada en la base de datos.

**TABLA 15.5**  
**Tipos de fallas, efectos y frecuencia**

Tipo	Efecto	Frecuencia
Detectada por un programa	Local (una transacción)	La más frecuente
Terminación anormal	Local (una transacción)	Frecuencia moderada
Falla del sistema	Global (todas las transacciones activas)	No es frecuente
Falla del dispositivo	Global (todas las transacciones activas y pasadas)	La menos frecuente

Los dos últimos tipos de fallas tienen consecuencias más serias, pero son mucho menos comunes. Una falla del sistema es una terminación anormal del sistema operativo. Una falla del sistema operativo afecta todas las transacciones que se están ejecutando. Una falla del dispositivo, como el colapso de un disco, afecta todas las transacciones en ejecución y todas aquellas que ya se realizaron y cuyo trabajo está grabado en el disco. La recuperación de una falla del dispositivo puede tardar horas, mientras que un colapso del sistema puede corregirse en minutos.

### 15.3.2 Herramientas de recuperación

El administrador de recuperaciones utiliza la redundancia y el control del tiempo de las escrituras en una base de datos para restaurarla después de una falla. Tres de las herramientas que estudiamos en esta sección son formas de redundancia: registro de transacciones, punto de revisión y respaldo de base de datos. La última herramienta (escritura forzada) permite al administrador de recuperaciones tener el control del momento en que se graban las escrituras a una base de datos. Esta sección explica la naturaleza de estas herramientas, mientras que la siguiente explica cómo se utilizan en los procesos de recuperación.

#### *Registro de transacciones*

##### **registro de transacciones**

tabla que contiene el historial de los cambios en la base de datos. El administrador de recuperaciones utiliza la tabla de registro para recuperar la base de datos de una falla.

Un registro de transacciones proporciona el historial de los cambios en la base de datos. Cada cambio realizado a una base queda asentado también en el registro. El registro es una tabla oculta que no está disponible para los usuarios normales. Un registro típico (tabla 15.6) contiene un número de secuencia de registros lógico (LSN; *log sequence number*), un identificador de transacciones, una acción en la base de datos, la hora, un identificador de fila, un nombre de columna y valores (antiguos y nuevos). Para insertar operaciones, el nombre de columna \* indica todas las columnas con el nuevo valor, conteniendo una fila completa de valores. En ocasiones, los valores antiguos y nuevos se conocen como imágenes de antes y después, respectivamente. Si la acción en la base de datos es insertar, el registro sólo contiene los nuevos valores. De modo similar, si la acción es eliminar, el registro sólo contiene los antiguos valores. Además de las acciones de insertar, actualizar y eliminar, se crean registros para el inicio y el final de una transacción.

El administrador de recuperaciones puede realizar dos operaciones en el registro. En una operación deshacer, la base de datos regresa a un estado previo sustituyendo el valor antiguo para cualquier valor que esté guardado en la base de datos. En una operación de rehacer, el componente de recuperación se restablece en un estado nuevo, sustituyendo el valor nuevo con cualquiera que esté guardado en la base de datos. Para deshacer (rehacer) una transacción, la operación deshacer (rehacer) se aplica a todos los registros de una transacción específica, excepto por los registros de inicio y realización.

Un registro puede representar un almacenamiento adicional considerable. En un ambiente de transacciones en volúmenes muy altos, es posible que se generen 100 gigabytes de registros al día. Debido al volumen, muchas organizaciones tienen un registro en línea guardado en disco y un registro en archivo guardado en cinta o unidad óptica. Por lo regular, el registro en línea se divide en dos partes (actual y siguiente) para administrar el espacio de almacenamiento en línea. La integridad del registro es crucial en vista de su función en el proceso de recuperación. Los DBMS empresariales pueden conservar registros redundantes para ofrecer un procesamiento sin interrupciones en caso de una falla en un registro.

#### *Control*

El propósito de un control es reducir el tiempo de recuperación de las fallas. A intervalos, se escribe un registro de control en el registro para guardar todas las transacciones activas. Además,

**TABLA 15.6 Ejemplo de registro de transacciones para una transacción en cajero automático**

LSN	TransNo	Acción	Hora	Tabla	Fila	Columna	Antiguo	Nuevo
1	101001	START	10:29					
2	101001	UPDATE	10:30	Acct	10001	AcctBal	100	200
3	101001	UPDATE	10:30	Acct	15147	AcctBal	500	400
4	101001	INSERT	10:32	Hist	25045	*		<1002, 500,-.-.->
5	101001	COMMIT	10:33					

todos los búfers de registro y algunos de la base de datos se escriben en el disco. En el momento de reiniciar, el administrador de recuperaciones depende del registro del control y de su conocimiento sobre las escrituras en el registro y las páginas de la base de datos para reducir la cantidad de trabajo de reinicio.

El intervalo entre revisiones se define como el periodo entre controles, y puede expresarse como un tiempo (como cinco minutos) o un parámetro de tamaño (como el número de transacciones realizadas, el número de páginas del registro o el número de páginas de la base de datos). El intervalo entre controles es un parámetro de diseño. Un intervalo breve reduce el trabajo de reinicio, pero provoca más trabajo adicional para registrar los controles. Un intervalo prolongado reduce el trabajo de revisión adicional, pero aumenta el trabajo al reiniciar. Un intervalo típico entre controles es de 10 minutos para volúmenes de transacciones elevados.

El registro de un control puede representar una interrupción considerable en el procesamiento de las transacciones, ya que toda la actividad cesa cuando ocurre una revisión. Durante el control no es posible empezar ninguna transacción nueva y las transacciones existentes no pueden iniciar operaciones nuevas. La duración de la interrupción depende del tipo de control utilizado. En un control consistente con el caché, las páginas del búfer (páginas de registro y algunas páginas sucias en la base de datos) que permanecen en la memoria se escriben en el disco y luego se escribe también el registro del control. Una página es sucia si fue cambiada por una transacción.

Para reducir la interrupción provocada por los controles consistentes con el caché, algunos DBMS soportan controles confusos o incrementales. En un control confuso, el administrador de recuperaciones sólo escribe las páginas del búfer desde el control anterior. La mayor parte de estas páginas ya deben estar escritas en el disco antes del control. En el momento de reinicio, el administrador de recuperaciones utiliza los registros de los controles confusos más recientes. Por lo tanto, este tipo de controles comprende menos trabajo adicional que los que son consistentes con el caché, pero que requieren más trabajo de reinicio.

En un control incremental, ninguna página de la base de datos se escribe en el disco. En vez de ello, las páginas sucias de la base de datos se escriben en forma periódica en el disco en orden de antigüedad ascendente. En el momento del control, se graba la posición en el registro de la página de datos sucia más antigua con el fin de ofrecer un punto de inicio para la recuperación. La cantidad de trabajo en el reinicio se puede controlar con la frecuencia de escritura de las páginas de datos sucias.

### *Escritura forzada*

#### **escritura forzada**

la habilidad de controlar el momento en que los datos se transfieren a un almacenamiento no volátil

la habilidad de controlar el momento en que los datos se transfieren a un almacenamiento no volátil. Esta capacidad es fundamental para el administrador de recuperaciones.

La habilidad de controlar el momento en que los datos se transfieren a un almacenamiento no volátil se conoce como escritura forzada. Sin esta capacidad, la recuperación no es posible. La escritura forzada significa que el DBMS, y no el sistema operativo, controla el momento de escribir los datos en un almacenamiento no volátil. Por lo general, cuando un programa emite un comando de escritura, el sistema operativo pone los datos en un búfer. Para mayor eficiencia, los datos no se escriben en el disco hasta que el búfer está lleno. Por lo regular, existe alguna demora entre la llegada de los datos al búfer y la transferencia de éste al disco. Con la escritura forzada, el sistema operativo permite al DBMS controlar el momento en que el búfer se escribe en el disco.

El administrador de recuperaciones utiliza la escritura forzada en el momento del control y al final de una transacción. En el control, además de insertar un registro de control, se tiene que escribir en el disco todo el registro y algunos búfers de la base de datos. Esta escritura forzada puede representar un trabajo adicional considerable en el proceso del control. Al final de una transacción, el administrador de recuperaciones escribe cualquier registro de la transacción que queda en la memoria.

### *Respaldo de base de datos*

Un respaldo es una copia de todo un disco o parte de éste. El respaldo se utiliza cuando está dañado el disco que contiene la base de datos o el registro. Por lo general, el respaldo se hace en cinta magnética porque es menos costosa y más confiable que el disco. Es preciso realizar en forma periódica un respaldo tanto de la base de datos como del registro. Para ahorrar tiempo, la mayor parte de los respaldos programados incluyen copias de todo el contenido de un disco y respaldos incrementales más frecuentes para copiar sólo las partes que cambiaron.

### 15.3.3 Procesos de recuperación

El proceso de recuperación depende del tipo de falla. La recuperación de una falla del dispositivo es sencilla, pero puede tomar mucho tiempo, como explicamos a continuación:

- Se recupera la base de datos del respaldo más reciente.
- Luego, el administrador de recuperaciones aplica el operador `rehacer` a todas las transacciones realizadas después del respaldo. Como el respaldo puede ser de varias horas o días atrás, es preciso consultar el registro para restaurar las transacciones realizadas después de éste.
- El proceso de recuperación termina con el reinicio de las operaciones incompletas.

Para las fallas locales y del sistema, el proceso de recuperación depende del momento en que los cambios a la base de datos se graban en el disco. Los cambios en la base de datos pueden grabarse antes de la realización (actualización inmediata) o después de ésta (actualización diferida). La cantidad de trabajo y el uso de las operaciones de registro (`deshacer` y `rehacer`) dependen del momento en que se realicen las actualizaciones a la base de datos. El resto de esta sección describe los procesos de recuperación para las fallas locales y del sistema en cada escenario.

#### *Actualización inmediata*

##### **planteamiento de actualización inmediata**

las actualizaciones a la base de datos se escriben en el disco en el momento en que ocurren, pero después de las actualizaciones correspondientes al registro. Para restaurar una base de datos, quizás se necesiten operaciones de `deshacer` y `rehacer`.

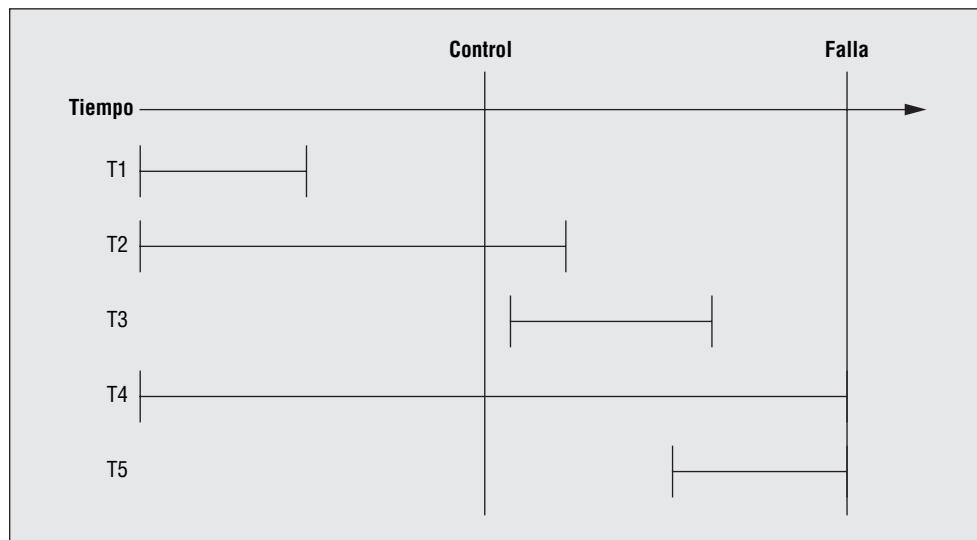
En el planteamiento de actualización inmediata, las actualizaciones a la base de datos se escriben en el disco cuando ocurren. Las escrituras a la base de datos también se realizan en el momento del punto de revisión, dependiendo de su algoritmo. Las escrituras a la base de datos deben ocurrir después de las escrituras a los registros correspondientes. Este uso del registro se conoce como protocolo de escritura previa en el registro. Si los registros se escribieran después de sus correspondencias en la base de datos, no sería posible una recuperación en caso de que ocurriera una falla entre el momento de escribir los registros en la base de datos y en el registro. Para asegurar el protocolo de escritura previa en el registro, el administrador de recuperaciones mantiene en un búfer una tabla de números de secuencia de registros para cada página de la base de datos. Una página de la base de datos no puede escribirse en el disco si su número de secuencia de registro asociado es mayor que el número de secuencia del último registro escrito en el disco.

La recuperación de una falla local es muy sencilla porque se ve afectada una sola transacción. Todos los registros de la transacción se encuentran al buscar los antecedentes del registro. Después, se aplica la operación `deshacer` a cada registro de la transacción. Si ocurre una falla durante el proceso de recuperación, vuelve a aplicarse la operación `deshacer`. El efecto de aplicar el operador `deshacer` varias veces es el mismo que si se aplica una sola vez. Al terminar las operaciones `deshacer`, el administrador de recuperaciones puede ofrecer al usuario la oportunidad de reiniciar la transacción cancelada.

La recuperación de una falla del sistema es más difícil porque todos los usuarios activos se ven afectados. Para ayudarle a entender la recuperación de una falla del sistema, la figura 15.10 muestra el progreso de un número de transacciones con respecto al tiempo asignado, el control más reciente y la falla. Cada transacción representa una clase de transacciones. Por ejemplo, la clase de transacciones T1 representa las transacciones que empezaron y terminaron antes del punto de revisión (y la falla). No es posible ningún otro tipo de transacción.

El planteamiento de actualización inmediata puede comprender operaciones de `deshacer` y `rehacer`, como resume la tabla 15.7. Para entender la cantidad de trabajo necesaria, recuerde que los registros son estables en el momento del control, y que el final de la transacción y los cambios a la base de datos también lo son. Aunque otras escrituras a la base de datos ocurran cuando el búfer se llena, el momento en que se presentan es impredecible. Las transacciones T1 no requieren de ningún trabajo porque los cambios al registro y a la base de datos son estables antes de la falla. Las transacciones T2 deben volverse a realizar desde el control porque sólo los cambios a la base de datos previos al mismo son estables. Las transacciones T3 deben volverse a realizar por completo porque no se garantiza que los cambios a la base de datos sean estables, aun cuando algunos pueden estar grabados en el disco. Las transacciones T4 y T5 deben deshacerse por completo porque, ya que es probable que algunos cambios a la base de datos se hayan grabado en el disco después del control.

**FIGURA 15.10**  
Cronograma de transacciones



**TABLA 15.7**  
Resumen del trabajo de reinicio para el planteamiento de actualización inmediata

Clase	Descripción	Trabajo de reinicio
T1	Terminada antes del control	Ninguno
T2	Iniciada antes del control; terminada antes de la falla	Rehacer hacia adelante desde el control más reciente
T3	Iniciada después del control; terminada antes de la falla	Rehacer hacia adelante desde el control más reciente
T4	Iniciada antes del control; todavía sin terminar	Deshacer hacia atrás desde el registro más reciente
T5	Iniciada después del control; todavía sin terminar	Deshacer hacia atrás desde el registro más reciente

**TABLA 15.8**  
Resumen del trabajo de reinicio para el planteamiento de actualización diferida

Clase	Descripción	Trabajo de reinicio
T1	Terminada antes del control	Ninguno
T2	Iniciada antes del control; terminada antes de la falla	Rehacer hacia adelante desde el primer registro
T3	Iniciada después del control; terminada antes de la falla	Rehacer hacia adelante desde el primer registro
T4	Iniciada antes del control; todavía sin terminar	Ninguno
T5	Iniciada después del control; todavía sin terminar	Ninguno

### Actualización diferida

**planteamiento de actualización diferida**  
las actualizaciones a una base de datos sólo se escriben después de que se realiza una transacción. Para restaurar una base de datos, sólo se utilizan las operaciones de rehacer.

En el planteamiento de actualización diferida, las actualizaciones a la base de datos se escriben en el disco sólo después de que se realiza una transacción. Ninguna escritura ocurre en el momento del control, excepto las transacciones que ya se realizaron. La ventaja del planteamiento de la actualización diferida es que no es necesaria ninguna operación deshacer. Sin embargo, tal vez sea necesario realizar más operaciones rehacer que en el planteamiento de actualización inmediata.

En el planteamiento de actualización diferida, las fallas locales se manejan sin ningún trabajo de reinicio. Como no ocurre ningún cambio a la base de datos sino hasta después de que se realiza una transacción, ésta se cancela sin ningún trabajo de deshacer. Por lo general, el administrador de recuperaciones da al usuario la opción de reiniciar la transacción.

Las fallas del sistema también pueden manejarse sin ninguna operación deshacer, como ilustra la tabla 15.8. Las transacciones T4 y T5 (todavía sin realizar) no requieren de operaciones deshacer porque no se escribe ningún cambio a la base de datos sino hasta que se realiza una transacción. Las transacciones T2 y T3 (realizadas después del control) no requieren de

operaciones deshacer porque no se sabe si todos los cambios a la base de datos son estables. Las transacciones T2 (iniciadas antes del control) deben rehacerse desde el primer registro, en lugar de hacerlo desde el control, como en el planteamiento de actualización inmediata. Por lo tanto, la actualización diferida requiere de más trabajo de reinicio para las transacciones T2, que el planteamiento de la actualización inmediata. Sin embargo, la actualización diferida no requiere de trabajo de reinicio para las transacciones T4 y T5, mientras que la inmediata debe deshacer este tipo de transacciones.

### *Ejemplo de recuperación*

Aunque las tablas 15.7 y 15.8 ilustran el tipo de trabajo de reinicio necesario, no muestran la secuencia de las operaciones del registro. Para ayudarle a entender las operaciones del registro que se generan en el momento del reinicio, la tabla 15.9 muestra un ejemplo de registro que incluye los registros del control. La acción del control comprende una lista de transacciones activas en el momento del control. Para simplificar el proceso de reinicio, se toman en cuenta los controles consistentes con el caché.

La tabla 15.10 muestra las operaciones de registro para el planteamiento de actualización inmediata. Este planteamiento comienza en la etapa de regreso al estado original. En el paso 1, el administrador de recuperaciones agrega la transacción 4 a la lista de transacciones sin realizar y aplica el operador deshacer a LSN 20. De modo similar, en el paso 4, el administrador de recuperaciones agrega la transacción 3 a la lista de transacciones sin realizar y aplica el operador deshacer a LSN 17. En el paso 5, el administrador de recuperaciones agrega la transacción 2 a la lista de transacciones sin realizar porque el registro contiene la acción COMMIT. En el paso 6, el administrador de recuperaciones no emprende ninguna acción porque las operaciones rehacer se van a aplicar en las etapas siguientes. En el paso 11, termina la etapa hacia atrás porque se encuentra el registro START para la última transacción en la lista de transacciones sin realizar. En la etapa hacia delante, el administrador de recuperaciones utiliza las operaciones rehacer para cada registro de las transacciones activas.

La tabla 15.11 presenta las operaciones de registro para el planteamiento de actualización diferida. El administrador de recuperaciones empieza por leer los registros hacia atrás como en el planteamiento de actualización inmediata. En el paso 1, el administrador ignora los registros 20 a 17, porque comprenden transacciones que no se realizaron antes de la falla. En los pasos 2 y 4, el administrador observa que las transacciones 1 y 2 ya se realizaron y será necesario rehacerlas durante la etapa hacia delante. En el paso 3, el administrador no emprende ninguna acción

**TABLA 15.9**  
**Ejemplo de registro de transacciones**

LSN	TransNo	Acción	Hora	Tabla	Fila	Columna	Antiguo	Nuevo
1	1	START	10:29					
2	1	UPDATE	10:30	Acct	10	Bal	100	200
3		CKPT(1)	10:31					
4	1	UPDATE	10:32	Acct	25	Bal	500	400
5	2	START	10:33					
6	2	UPDATE	10:34	Acct	11	Bal	105	205
7	1	INSERT	10:35	Hist	101	*		<1 400, ... >
8	2	UPDATE	10:36	Acct	26	Bal	100	200
9	2	INSERT	10:37	Hist	102	*		<2 200, ... >
10	3	START	10:38					
11	3	UPDATE	10:39	Acct	10	Bal	100	200
12		CKPT(1,2,3)	10:40					
13	3	UPDATE	10:41	Acct	25	Bal	500	400
14	1	COMMIT	10:42					
15	2	UPDATE	10:43	Acct	29	Bal	200	300
16	2	COMMIT	10:44					
17	3	INSERT	10:45	Hist	103	*		<3 400, ... >
18	4	START	10:46					
19	4	UPDATE	10:47	Acct	10	Bal	100	200
20	4	INSERT	10:48	Hist	104	*		<3 200, ... >

**TABLA 15.10**

**Trabajo de reinicio con el uso del planTEAMIENTO DE ACTUALIZACIÓN INMEDIATA**

Número de paso	LSN	Acciones
1	20	Deshacer; agregar 4 a la lista de transacciones no realizadas
2	19	Deshacer
3	18	Eliminar 4 de la lista de transacciones no realizadas
4	17	Deshacer; agregar 3 a la lista de transacciones no realizadas
5	16	Agregar 2 a la lista de transacciones no realizadas
6	15	Sin acción
7	14	Agregar 1 a la lista de transacciones no realizadas
8	13	Deshacer
9	12	Indicar que 3 sigue sin realizar
10	11	Deshacer
11	10	Eliminar 3 de la lista de transacciones no realizadas
12		Etapa hacia delante; empezar a leer el registro en el punto de revisión más reciente (LSN = 12)
13	14	Eliminar 1 de la lista de transacciones realizadas
14	15	Rehacer
15	16	Eliminar 2 de la lista de transacciones realizadas; detenerse porque la lista de transacciones realizadas está vacía

**TABLA 15.11**

**Trabajo de reinicio con el uso del planTEAMIENTO DE ACTUALIZACIÓN DIFERIDA**

Número de paso	LSN	Acciones
1	20, 19, 18, 17	Ninguna porque las transacciones no se pueden completar
2	16	Agregar 2 a la lista de transacciones realizadas e incompletas
3	15	Ninguna; rehacer durante la etapa hacia delante
4	14	Agregar 1 a la lista de transacciones realizadas e incompletas
5	13	Ninguna porque la transacción no se puede completar
6	12	Indicar que no se encontraron registros START para 1 y 2
7	11, 10	Ninguna porque la transacción no se puede completar
8	9, 8, 7, 6	Ninguna; rehacer durante la etapa hacia delante
9	5	Eliminar 2 de la lista de transacciones incompletas (se encontró el registro START)
10	4	Ninguna; rehacer durante la etapa hacia delante
11	3	Indicar que no se encontró registro START para 1
12	2	Ninguna. Rehacer durante la etapa hacia delante
13	1	Eliminar 1 de la lista de transacciones incompletas; empezar la etapa hacia delante
14	2	Rehacer
15	4	Rehacer
16	6	Rehacer
17	7	Rehacer
18	8	Rehacer
19	9	Rehacer
20	14	Eliminar 1 de la lista de transacciones realizadas
21	15	Rehacer
22	16	Eliminar 2 de la lista de transacciones realizadas; terminar la etapa hacia delante

porque más adelante se realizará una operación rehacer durante la etapa hacia adelante. La etapa hacia atrás continúa hasta encontrar registros START para todas las transacciones realizadas. En la etapa hacia adelante, el administrador de recuperaciones rehace las operaciones para cada acción de las transacciones en la lista de transacciones realizadas. La etapa hacia delante termina cuando el administrador de recuperaciones encuentra el registro COMMIT de la última transacción en la lista de transacciones realizadas.

#### *Proceso de recuperación en Oracle 10g*

Para ilustrar el proceso de recuperación utilizado en un DBMS empresarial, se presentan las selecciones del administrador de recuperaciones en Oracle 10g. Oracle utiliza el proceso de

actualización inmediata con controles incrementales. En el momento control no ocurre ninguna escritura en la base de datos, ya que éstas se escriben en forma periódica en el disco en orden de antigüedad ascendente. Para identificar el punto de inicio, se escribe en el disco el número de secuencia de registro correspondiente a la página sucia más antigua de la base de datos.

Los controles incrementales comprenden un intercambio entre la frecuencia de escritura de páginas sucias en la base de datos y el tiempo de reinicio. Las escrituras más frecuentes hacen más lento el caudal de procesamiento de las transacciones, pero reducen el tiempo de reinicio. Para controlar este intercambio, Oracle proporciona un parámetro conocido como tiempo medio de recuperación (MTTR; *Mean Time to Recover*), definido como el tiempo esperado (en segundos) para recuperarse de una falla en el sistema. La reducción de este parámetro da lugar a escrituras más frecuentes en las páginas de una base de datos. Para ayudar al diseñador a establecer el parámetro MTTR, Oracle ofrece el consejero MTTR, que elige los valores del parámetro con distintas cargas de trabajo. Para vigilar el proceso de recuperación, Oracle proporciona una vista de diccionario dinámico que contiene los detalles sobre las condiciones del proceso de recuperación.

## 15.4 Aspectos del diseño de transacciones

---

Con los servicios de recuperación y concurrencia, puede sorprender el hecho de que el diseñador de transacciones todavía tenga que tomar decisiones importantes de diseño. El diseñador puede ser un administrador de bases de datos, un programador o un programador asesorado por un administrador de bases de datos. Las decisiones de diseño pueden tener un impacto significativo en el desempeño del procesamiento de las transacciones. Los conocimientos acerca de los detalles del control de concurrencia y la recuperación lo convierten en un mejor diseñador de transacciones. Esta sección describe las decisiones disponibles para que los diseñadores de transacciones mejoren su desempeño.

### 15.4.1 Límite de transacción y puntos decisivos

Un diseñador de transacciones desarrolla una aplicación para realizar parte del procesamiento de una base de datos. Por ejemplo, puede desarrollar una aplicación para permitir que un usuario retire dinero de un cajero automático, pida un producto o se registre en un curso. Para crear la aplicación, el diseñador utiliza la transacción que define los enunciados de SQL y los servicios de control de concurrencia y recuperación del DBMS. El diseñador tiene varias alternativas acerca del lugar donde va a usar los enunciados de SQL que definen las transacciones. Esta decisión se conoce como límite de transacción.

Por lo general, el diseñador de transacciones tiene la opción de realizar una transacción extensa que contiene todos los enunciados SQL, o dividir estos enunciados en varias transacciones menores. Por ejemplo, los enunciados SQL en la transacción del cajero automático pueden considerarse una transacción, como muestra la figura 15.1. Otra opción es hacer que cada enunciado SQL sea una transacción por separado. Cuando no se usan los enunciados de límite de transacción (START TRANSACTION y COMMIT), cada enunciado SQL es una transacción separada de manera predeterminada.

#### *Intercambios al elegir los límites de transacción*

Al elegir el límite de una transacción, el objetivo es minimizar su duración al tiempo que se garantiza la satisfacción de las restricciones críticas. Los DBMS están diseñados para transacciones de corta duración porque los candados pueden obligar a que otras transacciones tengan que esperar. La duración incluye no sólo el número de lecturas y escrituras en una base de datos, sino también el tiempo invertido en esperar las respuestas del usuario. Por lo general, el límite de una transacción no debe comprender la interacción con el usuario. En las transacciones realizadas en cajeros automáticos, sistemas de reservaciones en líneas aéreas y transacciones de pedidos de productos (figuras 15.1, 15.2 y 15.3, respectivamente), los enunciados START TRANSACTION y COMMIT pueden moverse para rodear sólo la parte SQL del pseudocódigo.

La duración no debe comprometer la revisión de restricciones. Como la revisión de restricciones debe ocurrir al final de una transacción, tal vez sea difícil revisar algunas restricciones

#### **Límite de transacción**

una decisión importante del diseño de transacciones, en la que una aplicación, que consiste en un grupo de enunciados SQL, se divide en una o más transacciones.

si la transacción se divide en transacciones más pequeñas. Por ejemplo, una restricción importante en las transacciones de contabilidad es que los débitos sean iguales que los créditos. Si los enunciados SQL para publicar un débito y un crédito se colocan en la misma transacción, el DBMS puede poner en práctica la restricción contable al final de la transacción. Si se colocan en transacciones separadas, la revisión de restricciones no puede ocurrir sino hasta que se realizan ambas transacciones.

### Puntos decisivos

Para entender los efectos de las decisiones sobre los límites de las transacciones, es preciso identificar los puntos decisivos. Recuerde que los puntos decisivos son datos comunes que varios usuarios tratan de cambiar al mismo tiempo. Si el límite de una transacción elimina (crea) un punto decisivo, tal vez el diseño sea adecuado (inapropiado).

Los puntos decisivos pueden clasificarse como independientes o dependientes del sistema. Los puntos decisivos independientes del sistema forman parte de una tabla que quizás varios usuarios quieran cambiar al mismo tiempo. Filas, columnas y tablas completas pueden ser puntos decisivos independientes del sistema. Por ejemplo, en la transacción de reservación en una línea aérea (figura 15.2), la columna de los asientos restantes de las filas de vuelos con gran demanda es un punto decisivo independiente del sistema. La columna de asientos restantes es un punto decisivo en cualquier DBMS.

Los puntos decisivos dependientes del sistema dependen del DBMS. Por lo general, éstos comprenden partes de la base de datos ocultas a los usuarios normales. Las páginas (registros físicos) que contienen filas de la base de datos o registros en el índice a menudo son puntos decisivos dependientes del sistema. Por ejemplo, algunos DBMS ponen candados a la página siguiente al insertar una fila en una tabla. Al insertar una nueva fila de historial en la transacción realizada en el cajero automático (figura 15.1), la página siguiente de la tabla del historial es un punto decisivo dependiente del sistema. En los DBMS que usan candados en el nivel de filas, no hay ningún punto decisivo. Por lo regular, también hay puntos decisivos con ciertas páginas de índice con acceso frecuente.

### Ejemplo de diseño del límite de una transacción

Para ilustrar la opción del límite de una transacción, los formularios jerárquicos ofrecen un contexto conveniente. Un formulario jerárquico representa una aplicación que lee y escribe en una base de datos. Por ejemplo, el formulario de registro de la base de datos de la universidad (figura 15.11) manipula la tabla *Registration* en el formulario principal y las tablas *Enrollment* y *Offering* en el subformulario. Al utilizar el formulario de registro para las inscripciones en los cursos, se inserta un registro en la tabla *Registration* después de completar el formulario principal. Al completar cada línea en el subformulario, se inserta una fila en la tabla *Enrollment*.

**FIGURA 15.11**

Ejemplo de formulario de registro

Offer No.	Course No.	Units	term	year	location	Time	Instructor
1234	IS320	4	FALL	2005	BLM302	10:30 AM	LEONARD VINCE

y se actualiza la columna *OffSeatsRemain* de la fila *Offering* asociada. Aunque el campo *OffSeatsRemain* no aparece en el subformulario, es preciso actualizarlo después de insertar cada línea en el subformulario.

Al diseñar un formulario jerárquico, el programador de bases de datos tiene tres opciones razonables para el límite de una transacción:

1. El formulario completo.
2. El formulario principal como una transacción y *todas* las líneas del subformulario como una segunda transacción.
3. El formulario principal como una transacción y *cada* línea del subformulario como transacciones separadas.

La tercera opción casi siempre es la preferida porque ofrece transacciones con menor duración. Sin embargo, la revisión de restricciones puede obligar a elegir las opciones 1 o 2. En el formulario de registro hay restricciones que comprenden un registro completo, como el número de horas mínimo para una ayuda financiera y los requisitos previos para un curso. Sin embargo, no es crucial revisar estas restricciones al final de una transacción. La mayoría de las universidades revisan estas restricciones posteriormente, antes de que empiece el siguiente periodo académico. Por tanto, la opción 3 es la mejor para el límite de una transacción.

Hay varios puntos decisivos comunes para cada opción del límite de transacción. La columna *OffSeatsRemain* en las populares filas *Offering* es un punto decisivo independiente del sistema en cualquier transacción que comprenda las líneas del subformulario. La columna *OffSeatsRemain* debe actualizarse después de cada inscripción. En algunos DBMS, la página siguiente en la tabla *Enrollment* es un punto decisivo dependiente del sistema. Después de cada línea del subformulario, se inserta una fila en la tabla *Enrollment*. Si el DBMS utiliza un candado en la página siguiente en lugar solamente de la nueva fila, todas las transacciones de la línea del subformulario deben obtener un candado exclusivo en el siguiente registro físico disponible. No obstante, si el DBMS puede poner un candado en el nivel de la fila, no hay ningún punto decisivo porque cada transacción inserta una fila diferente.

La opción 3 ofrece otra ventaja gracias a las posibilidades reducidas de un punto muerto. En las opciones 1 y 2, los puntos muertos son posibles porque las transacciones comprenden varias inscripciones a los cursos. Por ejemplo, una transacción podría obtener un candado en los ofrecimientos de comunicaciones de datos (IS470) y otra transacción en un ofrecimiento de base de datos (IS480). La primera transacción podría esperar un candado en el ofrecimiento IS480, y la segunda un candado en el ofrecimiento IS470. La opción 3 no tendrá puntos muertos si todas las transacciones obtienen los puntos decisivos en el mismo orden en todas las transacciones. Por ejemplo, no ocurrirá un punto muerto si cada transacción obtiene primero un candado en la siguiente página disponible de *Enrollment* y luego obtiene un candado en una fila de *Offering*.

#### *Cómo evitar el tiempo de interacción de usuarios*

Otro aspecto del límite de transacciones es la interacción de usuarios. Por lo general, ésta debe tener lugar fuera de la transacción. La figura 15.12 muestra la transacción de reservación en

**FIGURA 15.12**  
Pseudocódigo para la transacción de reservación en una línea aérea rediseñada

```

Muestra bienvenida
Obtener preferencias de reserva
SELECT registro de vuelos de salida y regreso
If la reserva es aceptable then
    START TRANSACTION
    UPDATE asientos disponibles del registro de salida de vuelos
    UPDATE asientos disponibles del registro de llegada de vuelos
    INSERT registro de reserva
End If
On Error: ROLLBACK
COMMIT
Enviar recibo al consumidor

```

una línea aérea rediseñada para colocar la interacción de usuarios fuera de ella. La transacción rediseñada reducirá los tiempos de espera de los usuarios concurrentes porque los candados se mantienen durante menos tiempo. Sin embargo, es probable que ocurran efectos secundarios como resultado de la eliminación de la interacción de los usuarios. Con ésta fuera de la transacción, es probable que no haya un asiento libre en el vuelo, aun cuando al usuario se le acabe de informar sobre su disponibilidad. Un analista de bases de datos deberá vigilar la ocurrencia de este efecto secundario. En caso de que sea raro, el código de interacción debe permanecer fuera de la transacción. Si el efecto secundario ocurre con razonable frecuencia, quizás sea preferible conservar la interacción de usuarios como parte de la transacción.

### 15.4.2 Niveles de aislamiento

Si todos los candados en dos etapas se conservan hasta el final de la transacción, se evitan los tres problemas de control de concurrencia que describimos en la sección 15.2.2. Sin embargo, es probable que algunas transacciones no necesiten este nivel de control de concurrencia. El problema de la actualización perdida es el más serio y siempre debe prevenirse. Algunas transacciones pueden tolerar los conflictos provocados por los problemas de dependencia y recuperación inconsistente. Quizás las transacciones que no necesitan protegerse de estos problemas pueden liberarse más pronto de los candados y alcanzar una velocidad de ejecución más alta.

El nivel de aislamiento especifica el grado en el que una transacción se separa de las acciones de otras transacciones. Un diseñador de transacciones puede equilibrar el trabajo adicional del control de concurrencia con los problemas de interferencia, que se previenen especificando el nivel de aislamiento apropiado.

La tabla 15.12 resume los niveles de aislamiento de SQL:2003 según la duración y el tipo de candado. El nivel serializado evita todos los problemas de control de concurrencia, pero comprende mayor trabajo adicional y más tiempo de espera. Para evitar los problemas de control de concurrencia, se utilizan candados de predicado, todos a largo plazo (se conservan hasta el momento de la realización). Los candados de predicado que reservan filas especificadas por las condiciones en la cláusula WHERE son esenciales para evitar problemas de lecturas fantasma. El nivel de lectura repetible utiliza candados de predicado a corto plazo para evitar el resumen incorrecto y los problemas de lectura no repetibles. El nivel de realización de lectura utiliza candados compartidos a largo plazo para permitir mayor acceso concurrente. Sin embargo, sólo evita el problema de dependencia no realizado y el problema tradicional de actualización perdida. Como el nivel de lectura no realizado no utiliza candados, es apropiado para acceso de sólo lectura a una base de datos.

Un diseñador de transacciones puede especificar el nivel de aislamiento usando el enunciado SQL SET TRANSACTION, como muestra el ejemplo 15.1. El enunciado SET TRANSACTION casi siempre se coloca justo antes de un enunciado START TRANSACTION, con el fin de alterar los valores predeterminados para las transacciones. En el ejemplo 15.1, el nivel de aislamiento se establece en READ COMMITTED. Las otras palabras clave son SERIALIZABLE, REPEATABLE READ y READ UNCOMMITTED. Algunos fabricantes de DBMS no ofrecen soporte para todos estos niveles, mientras otros soportan niveles adicionales.

**TABLA 15.12 Resumen de los niveles de aislamiento**

Nivel	Candados exclusivos	Candados compartidos	Candados de predicado	Problemas permitidos
Lectura no realizada	Ninguna, porque es de sólo lectura	Ninguna	Ninguna	Sólo dependencias no realizadas porque las transacciones deben ser de sólo lectura
Lectura realizada	Largo plazo	Corto plazo	Ninguna	Actualizaciones escolares perdidas, resumen incorrecto, lecturas no repetibles
Lectura repetible	Largo plazo	Largo plazo	Lectura a corto plazo, escritura a largo plazo	Lecturas fantasma
Serializable	Largo plazo	Largo plazo	Largo plazo	Ninguno

**EJEMPLO 15.1  
(SQL:2003)****Enunciado SET TRANSACTION para establecer el nivel de aislamiento de una transacción**

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
START TRANSACTION
...
COMMIT
```

**FIGURA 15.13**  
Ejemplo del problema de actualización escolar perdida

Transacción A	Tiempo	Transacción B
Obtener candado S en SR	$T_1$	
Leer SR(10)	$T_2$	
Liberar candado S en SR	$T_3$	
Si $SR > 0$ , entonces $SR = SR - 1$	$T_4$	
	$T_5$	Obtener candado S en SR
	$T_6$	Leer SR(10)
	$T_7$	Liberar candado S en SR
	$T_8$	Si $SR > 0$ , entonces $SR = SR - 1$
Obtener candado X en SR	$T_9$	
Escribir SR(9)	$T_{10}$	
Realizar	$T_{11}$	
	$T_{12}$	Obtener candado X en SR
	$T_{13}$	Escribir SR(9)

En SQL:2003, SERIALIZABLE es el nivel de aislamiento predeterminado. Este nivel se recomienda para la mayor parte de las transacciones, porque el nivel REPEATABLE READ sólo ofrece una mejora menor en el desempeño. El nivel READ UNCOMMITTED se recomienda para transacciones de sólo lectura que pueden tolerar inconsistencias en la recuperación.

Aunque SQL:2003 ofrece SERIALIZABLE como nivel predeterminado, algunos fabricantes de DBMS, como Oracle y Microsoft SQL, utilizan READ COMMITTED como el nivel predeterminado. READ COMMITTED puede ser un nivel predeterminado peligroso, ya que permite una variación del problema de actualización perdida, conocida como actualización escolar perdida. La palabra *escolar* es irónica en el sentido de que el problema de la actualización escolar perdida difiere sólo un poco del problema de actualización tradicional perdida. La figura 15.13 ilustra el problema de actualización escolar perdida. La única diferencia esencial entre ambos problemas es que la transacción A se realiza antes de que la transacción B cambie los datos comunes. Por lo tanto, la actualización escolar perdida es un problema potencial serio que no debe permitirse en la mayoría de las transacciones.

### 15.4.3 Momento de imposición de la restricción de integridad

Además de establecer el nivel de aislamiento, SQL permite el control del momento de imposición de la restricción de integridad. En forma predeterminada, las restricciones se ponen en práctica inmediatamente después de cada enunciado INSERT, UPDATE y DELETE. Para la mayor parte de las restricciones, como llaves primarias y foráneas, es apropiado ponerlas en práctica de inmediato. Si se viola una restricción, el DBMS emite una operación de restauración al estado original de la transacción. El regreso al estado original restaura la base de datos a una condición consistente, ya que las propiedades ACID garantizan la consistencia al final de una transacción.

Para las restricciones complejas, quizás no sea apropiado ponerlas en práctica de inmediato. Por ejemplo, una restricción de la carga de trabajo de los profesores garantiza que cada maestro imparte entre tres y nueve unidades cada semestre. Si una transacción asigna una carga de trabajo completa, es necesario diferir la revisión de la restricción hasta el final de la transacción. Para este tipo de restricciones complejas es preciso especificar el momento de la restricción.

**revisión de restricción diferida**  
poner en práctica restricciones de integridad al final de una transacción en lugar de inmediatamente después de cada enunciado de manipulación. Las restricciones complejas pueden beneficiarse de la revisión diferida.

El momento de la restricción comprende tanto la definición de la restricción como la definición de la transacción. SQL proporciona una cláusula opcional para el momento de la restricción que se aplica a las restricciones de llaves primarias, las restricciones de llaves foráneas, las restricciones de unicidad, las restricciones de revisión y las afirmaciones. Por lo regular, un administrador de bases de datos utiliza la cláusula del momento de la restricción para aquellas que requieren de una revisión diferida. Las restricciones que nunca necesitan una revisión diferida no necesitan la cláusula del momento, ya que el valor predeterminado es NOT DEFERRABLE. La cláusula del momento define el grado en que una restricción puede diferirse, además de la forma de ponerla en práctica (diferida o inmediata), como muestran los ejemplos 15.2 y 15.3.

**EJEMPLO 15.2  
(SQL:2003)**
**Cláusula de momento para la afirmación *FacultyWorkLoad***

La restricción puede diferirse y la puesta en práctica se difiere.

```
CREATE ASSERTION FacultyWorkLoad
  CHECK (NOT EXISTS
    ( SELECT Faculty.FacSSN, OffTerm, OffYear
      FROM Faculty, Offering, Course
      WHERE Faculty.FacSSN = Offering.FacSSN
        AND Offering.CourseNo = Course.CourseNo
      GROUP BY Faculty.FacSSN, OffTerm, OffYear
      HAVING SUM(CrsUnits) < 3 OR SUM(CrsUnits) > 9 ) )
  DEFERRABLE INITIALLY DEFERRED
```

**EJEMPLO 15.3  
(SQL:2003)**
**Cláusula de momento para la afirmación *OfferingConflict***

La restricción puede diferirse y la puesta en práctica es inmediata.

```
CREATE ASSERTION OfferingConflict
  CHECK (NOT EXISTS
    ( SELECT O1.OfferNo
      FROM Offering O1, Offering O2
      WHERE O1.OfferNo <> O2.OfferNo
        AND O1.OffTerm = O2.OffTerm
        AND O1.OffYear = O2.OffYear
        AND O1.OffDays = O2.OffDays
        AND O1.OffTime = O2.OffTime
        AND O1.OffLocation = O2.OffLocation ) )
  DEFERRABLE INITIALLY IMMEDIATE
```

Para cada transacción, el diseñador puede especificar si las restricciones que lo permiten se difieren o se ponen en práctica de inmediato, utilizando el enunciado SET CONSTRAINTS. Por lo regular, este enunciado se coloca justo después del enunciado START TRANSACTION, como muestra el ejemplo 15.4. El enunciado START CONSTRAINTS no es necesario para las restricciones que pueden diferirse con la puesta en práctica predeterminada diferida. Por ejemplo, si la afirmación *FacultyWorkLoad* es diferida, no es necesario ningún enunciado SET CONSTRAINTS porque la puesta en práctica predeterminada es diferida.

La implementación de la parte del momento de la restricción de SQL es muy variable. La mayoría de los DBMS no ofrecen soporte para esta parte exactamente como se especifica en el estándar. Muchos DBMS tienen distintas sintaxis y extensiones de lenguaje propietario para el momento de las restricciones.

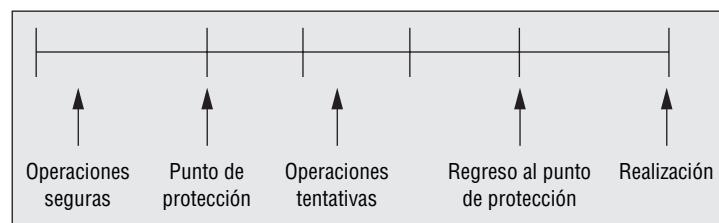
**EJEMPLO 15.4  
(SQL:2003)**

```

Enunciados SET CONSTRAINTS para varias transacciones
START TRANSACTION
SET CONSTRAINTS FacultyWorkLoad IMMEDIATE
...
COMMIT
START TRANSACTION
SET CONSTRAINTS OfferingConflict DEFERRED
...
COMMIT

```

**FIGURA 15.14**  
Flujo de una transacción con un punto de protección



#### 15.4.4 Puntos de protección

Algunas transacciones tienen acciones tentativas que pueden cancelarse mediante las acciones de los usuarios u otros eventos. Por ejemplo, el usuario puede cancelar un artículo en un pedido al descubrir que no lo hay en existencia. Como el enunciado ROLLBACK elimina todos los cambios a las transacciones, no puede utilizarse para borrar sólo el artículo cancelado si la transacción comprende un pedido completo. El diseñador de transacciones puede codificar los enunciados para eliminar de forma explícita las partes tentativas de una transacción, pero esta codificación puede ser tediosa y comprender un trabajo adicional excesivo.

SQL:2003 ofrece el enunciado SAVEPOINT para permitir el regreso parcial al punto de inicio de una transacción. Un diseñador utiliza la palabra clave SAVEPOINT seguida por el nombre del punto de protección para establecer un punto intermedio en una transacción. Para deshacer el trabajo desde un punto de protección en particular, es posible usar las palabras clave ROLLBACK TO SAVEPOINT seguidas por el nombre del punto de protección. La figura 15.14 ilustra el uso de un punto de protección. Por lo regular, un regreso parcial se utiliza en forma condicional dependiendo de la acción de un usuario o un evento externo.

Algunos DBMS empresariales usan también los puntos de protección de manera interna para resolver puntos muertos. En lugar de regresar una transacción completa al punto original, el DBMS la regresa a su último punto de protección. Un DBMS puede utilizar los puntos de protección implícitos después de cada enunciado SQL para reducir la cantidad de trabajo perdido.

### 15.5 Administración del flujo de trabajo

La administración de transacciones forma parte de un área mucho más extensa conocida como administración del flujo de trabajo. La administración del flujo de trabajo ofrece soporte para los procesos de negocios, tanto automatizados como realizados por un ser humano. En contraste, la administración de transacciones ofrece soporte para las propiedades del procesamiento automatizado de bases de datos. Esta sección presenta la administración del flujo de trabajo para ofrecer una perspectiva más amplia para la administración de transacciones. Esta sección describe primero los flujos de trabajo, un concepto más amplio que las transacciones de bases de datos. Luego, estudia las tecnologías para la administración del flujo de trabajo mostrando la importancia de la administración de transacciones.

### 15.5.1 Caracterización de los flujos de trabajo

#### **flujo de trabajo**

grupo de tareas estructuradas relacionadas para lograr un proceso de negocios.

Los flujos de trabajo soportan procesos de negocios tales como ofrecer un servicio telefónico, la obtención de un préstamo y pedir un producto. Los flujos de trabajo consisten en tareas que pueden realizar las computadoras (hardware y software), los seres humanos o una combinación de ambos. Por ejemplo, al dar un servicio telefónico, el software determina la hora de la cita para el servicio y actualiza una base de datos de horarios, mientras un técnico inspecciona la caja del teléfono para determinar si existe algún problema. Un flujo de trabajo define el orden en que se realizan las tareas, las condiciones para que se lleven a cabo y los resultados de las tareas realizadas. Por ejemplo, el hecho de prestar un servicio telefónico comprende el contacto inicial con el cliente, una visita opcional de servicio, la facturación y el cobro. Cada una de estas tareas puede tener condiciones en las cuales se prefieran, o puedan dar como resultado, acciones como la actualización de la base de datos y una invocación de otras tareas.

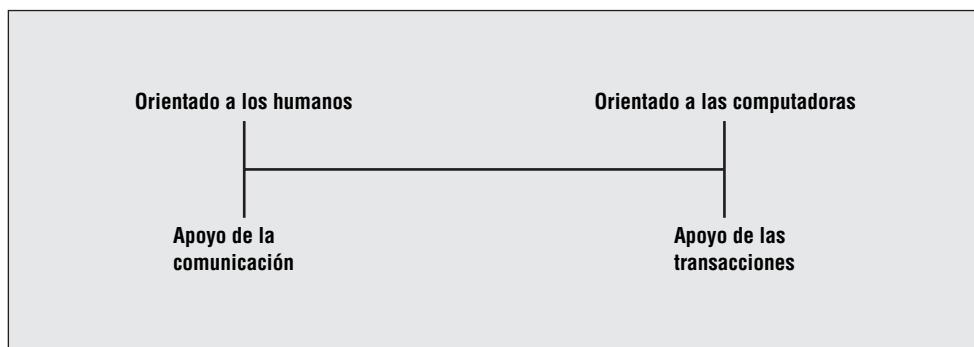
Existen muchos tipos de flujo de trabajo. Sheth, Georgakopoulos y Hornick (1995) clasificaron los flujos de trabajo como orientados a los humanos y a las computadoras, como ilustra la figura 15.15. En los flujos de trabajo orientados a los humanos, éstos ofrecen la mayor parte del sentido común para realizar el trabajo. La computadora tiene el papel pasivo de proporcionar los datos para facilitar las decisiones de las personas. Por ejemplo, al procesar un préstamo, a menudo los funcionarios de préstamos determinan las condiciones de éstos cuando los clientes no cumplen con los criterios estándar sobre el ingreso y la deuda. Quizá sea necesario consultar con fiadores y personal de crédito. Para apoyar los flujos de trabajo orientados a los humanos, puede ser útil el software de comunicación, como correo electrónico, chat y anotaciones en documentos. En las tareas orientadas a las computadoras, la computadora determina el procesamiento del trabajo. Por ejemplo, el software para una transacción en cajero automático determina si el cliente recibe dinero o si se rechaza su solicitud. La administración de transacciones es la tecnología clave para apoyar los flujos de trabajo orientados a los clientes.

Otra forma de clasificar los flujos de trabajo es por la complejidad de la tarea o por la estructura de ésta, como muestra la figura 15.16. La complejidad de la tarea comprende la dificultad de realizar tareas individuales. Por ejemplo, la decisión de otorgar un préstamo puede incluir un razonamiento complejo utilizando diversas variables. En contraste, el procesamiento de un pedido comprende la solicitud de datos acerca de éste por parte del cliente. La estructura de la tarea comprende las relaciones entre las tareas. Los flujos de trabajo en condiciones difíciles son muy estructurados. Por ejemplo, el procesamiento del cobro de un seguro puede tener condiciones relacionadas con la negación del pago, el litigio de éste y una investigación.

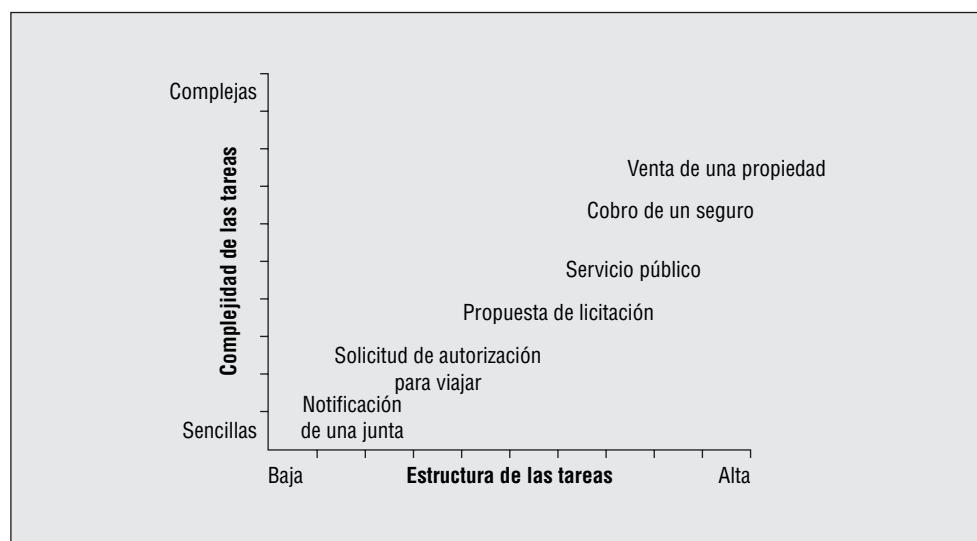
### 15.5.2 Tecnologías que permiten el flujo de trabajo

Para respaldar el concepto de un flujo de trabajo como el que estudiamos en la sección anterior, son importantes tres tecnologías: (1) administración de objetos distribuidos, (2) especificación del flujo de trabajo y (3) administración de transacciones personalizadas. Como describimos en la sección anterior, la administración de transacciones forma parte de la tercera tecnología. El resto de esta sección analiza cada una de las tecnologías.

**FIGURA 15.15**  
Clasificación del flujo de trabajo por desempeño de tareas



**FIGURA 15.16**  
Clasificación del flujo de trabajo por estructura y complejidad de tareas



### *Administración de objetos distribuidos*

Los flujos de trabajo pueden comprender muchos tipos de datos en lugares remotos. Por ejemplo, los datos pueden incluir fotos para el cobro de un seguro, rayos X respaldando un diagnóstico y la valuación de una propiedad para la solicitud de un préstamo. Por lo regular, los DBMS no manejan este tipo de datos, pero se ha desarrollado una nueva clase de DBMS, conocido como DBMS de objetos, que maneja diversos tipos de datos. El capítulo 18 describe esta nueva clase de DBMS.

Por lo general, además de los nuevos tipos de datos, la información no se guarda en un lugar y no la controlan distintos DBMS. Por ejemplo, para respaldar la solicitud de un préstamo, un funcionario de préstamos utiliza un reporte del buró de crédito, una valuación de un valuador certificado y lineamientos de procesamiento de préstamos de organismos gubernamentales. Puede ser difícil tener acceso y controlar los datos distribuidos. El capítulo 17 describe principios importantes para administrar este tipo de datos. También surgen dificultades porque los datos pueden estar controlados por diferentes sistemas, algunos de los cuales no ofrecen soporte para SQL.

### *Especificación e implementación del flujo de trabajo*

Para soportar los flujos de trabajo, la estructura de las tareas debe estar representada e implementada en forma apropiada. La representación de un flujo de trabajo comprende la identificación de las tareas y la especificación de las relaciones entre éstas. Una tarea compleja puede comprender una jerarquía de subtareas. Un flujo de trabajo complejo puede incluir muchas tareas con numerosas relaciones. Es posible utilizar restricciones, reglas y anotaciones gráficas para ilustrar el orden y la realización de las tareas. Un uso de las relaciones entre las tareas es definir las restricciones entre las transacciones. Por ejemplo, la especificación del flujo de trabajo puede indicar que a un estudiante se le debe negar la ayuda financiera a menos que se inscriba a un número mínimo de horas hasta una fecha específica.

Después de especificar un flujo de trabajo, es preciso implementarlo con eficiencia. La implementación puede comprender tipos muy diversos de hardware, software y personas. Un reto importante consiste en hacer que los distintos componentes se comuniquen con eficiencia. La optimización de los flujos de trabajo a través de la reingeniería se ha convertido en una preocupación importante en muchas organizaciones. La optimización puede comprender la eliminación de tareas duplicadas y el incremento del trabajo paralelo.

Han sido desarrollados muchos sistemas de software para soportar la especificación del flujo de trabajo. La función principal del software que lleva el nombre de administración de flujo de trabajo es soportar la especificación y la implementación del flujo de trabajo.

### *Administración de transacciones personalizadas*

Las primeras secciones de este capítulo describen el soporte de los DBMS a las transacciones ACID. Las propiedades ACID son muy importantes y los DBMS ofrecen un amplio soporte para éstas. Sin embargo, para soportar los flujos de trabajo, quizás estas propiedades no sean suficientes. La siguiente lista identifica las desventajas de las transacciones ACID tradicionales para la administración de flujos de trabajo:

- Algunos flujos de trabajo comprenden tareas de larga duración por la interacción del usuario. Es probable que la administración de transacciones tradicional no funcione bien para estas transacciones de conversación.
- Algunas tareas comprenden subtareas y cambian la idea de la atomicidad. Se ha propuesto la idea de las transacciones empaquetadas (transacciones dentro de transacciones) para tareas con estructuras complejas.
- Es probable que algunas tareas estén realizadas por sistemas heredados que no ofrecen soporte para las propiedades ACID.
- Algunos flujos de trabajo requieren que las tareas se deshagan antes de terminarlas. En los sistemas de contabilidad, es común que las transacciones de compensación corrijan los errores. Por ejemplo, la devolución de un producto defectuoso elimina el efecto del pedido original.

En la actualidad, ciertos DBMS ofrecen soporte para algunas de estas extensiones. Por ejemplo, Microsoft SQL Server ofrece transacciones empaquetadas para soportar las transacciones que residen en los procedimientos. Oracle ofrece transacciones autónomas para permitir la interrupción de una transacción por parte de otra. Además, SQL:2003 ofrece puntos seguros, de modo que el enunciado ROLLBACK puede deshacer sólo una parte de la transacción. Esta extensión reduce la cantidad de trabajo perdido cuando falla una transacción larga.

Para un soporte más completo de la administración del flujo de trabajo, es necesario personalizar la administración de las transacciones de acuerdo con los requisitos del flujo de trabajo. Las propiedades de las transacciones con soporte para un flujo de trabajo deben formar parte de la especificación de este último y no confundirse con el software que ofrece soporte para la administración del flujo de trabajo. Los DBMS que ofrecen soporte para la administración de flujos de trabajo deben ser más flexibles. Los DBMS pueden soportar distintos tipos de transacciones o incluso permitir la especificación de las propiedades de las transacciones. Incluso es posible utilizar el procesamiento en los DBMS para soportar características como las transacciones de compensación. La mayor parte de los DBMS necesitan una extensión importante para soportar la administración de transacciones personalizadas.

## Reflexión final

Este capítulo describió el concepto de las transacciones de bases de datos, los servicios que prestan los DBMS para soportar las transacciones y las habilidades que deben tener los diseñadores de transacciones. Una transacción es una unidad de trabajo identificada por el usuario con varias lecturas y escrituras en una base de datos. Para definir una transacción, presentamos varios enunciados SQL nuevos, entre ellos START TRANSACTION, COMMIT y ROLLBACK. Los DBMS garantizan que las transacciones sean atómicas (todo o nada), consistentes (satisfacen las restricciones de integridad después de su realización), aisladas (no interfieren con usuarios concurrentes) y duraderas (sobreviven a las fallas). Para garantizar estas propiedades de las transacciones, los DBMS ofrecen servicios de control de concurrencia (hacer que una base de datos parezca un sistema para un solo usuario) y administración de recuperaciones (restaurar automáticamente una base de datos después de una falla). Estos servicios no son gratis, ya que consumen grandes cantidades de recursos de computación y aumentan el precio de compra de un DBMS.

Aunque los servicios de concurrencia y recuperación que ofrece un DBMS sean transparentes, es preciso conocer algunos detalles. El conocimiento de estos servicios le ayudará a distribuir los recursos de cómputo, seleccionar un DBMS que ofrezca el nivel apropiado de soporte para las transacciones y diseñar transacciones eficientes. Para el control de concurrencias,

es preciso entender el objetivo, los problemas de interferencia y el protocolo de candados en dos etapas. Para la administración de las recuperaciones, debe entender la clase de fallas, el almacenamiento redundante necesario para la recuperación y la cantidad de trabajo para restaurar una base de datos después de una falla.

Para aplicar sus conocimientos sobre administración de transacciones, este capítulo demostró los principios del diseño de transacciones. La elección más importante en el diseño de transacciones es la selección de un límite para éstas. El objetivo de elegir un límite para las transacciones es minimizar la duración sujeta a la necesidad de revisar las restricciones. Las restricciones críticas, como el débito-crédito en los sistemas de contabilidad, pueden hacer que una aplicación permanezca como una sola transacción en lugar de dividirse en transacciones menores. Asimismo, es posible acortar el límite de la transacción eliminando la interacción del usuario. Otras decisiones importantes comprenden el nivel de aislamiento, el momento de poner en práctica las restricciones y los puntos seguros para un regreso parcial al estado original. A lo largo del capítulo se mostró la sintaxis de SQL para estos elementos.

Como diseñador de transacciones, debe recordar que éstas constituyen sólo un tipo de soporte para el trabajo en una organización. La administración del flujo de trabajo se ocupa de aspectos que van más allá de la administración de transacciones, como las dependencias entre éstas, los distintos tipos de transacciones y la anotación del trabajo.

Este capítulo describió los servicios que ofrece un DBMS para soportar el procesamiento de transacciones, el lado operativo de las bases de datos. Otros capítulos en la parte 7 estudian los servicios para soportar otro tipo de procesamiento de bases de datos, como el almacén de datos en el capítulo 16. Debe comparar los requisitos para soportar el procesamiento de transacciones para la toma de decisiones operativas con los requisitos para soportar los almacenes de datos para tomar decisiones tácticas y estratégicas.

## Revisión de conceptos

- Transacciones que contienen un grupo de lecturas y escrituras a la base de datos especificado por el usuario.
- Enunciados SQL para definir las transacciones: START TRANSACTION, COMMIT, ROLLBACK y SAVEPOINT.
- Propiedades ACID de las transacciones: atómicas, consistentes, aisladas y duraderas.
- Servicios transparentes para ocultar los detalles internos de la administración de transacciones.
- Control de concurrencia para soportar el uso simultáneo de una base de datos.
- Administración de recuperaciones para restaurar una base de datos a un estado consistente después de una falla.
- Objetivo del control de concurrencia: maximizar el caudal de procesamiento de las transacciones, al tiempo que evita problemas de interferencia.
- Interferencia en puntos decisivos, datos comunes manipulados por usuarios concurrentes.
- Tipos de problemas de interferencia: actualización perdida, dependencia sin realizar, recuperación inconsistente.
- Administrador del control de concurrencia para otorgar, liberar y analizar candados.
- Etapas de crecimiento y reducción de los candados en dos etapas (2PL).
- Resolución de puntos muertos con detección de puntos muertos y tiempo fuera, además del reinicio de las transacciones.
- Planteamientos de control de concurrencia optimistas cuando la interferencia es rara.
- Almacenamiento volátil contra no volátil.
- Efecto de las fallas locales, del sistema y de los medios.
- Escritura forzada para controlar el tiempo de las escrituras en la base de datos.
- Almacenamiento redundante para la recuperación: registro, control, respaldo.
- Intercambio en la frecuencia de los puntos de revisión: caudal de procesamiento de transacciones contra tiempo de reinicio.

- Cantidad de trabajo de reinicio en los planteamientos de actualización inmediata y la recuperación de actualización diferida.
- Selección de un límite de transacciones para minimizar la duración, al tiempo que se ponen en práctica restricciones críticas de integridad.
- Eliminar la interacción del usuario para reducir la duración de las transacciones.
- Identificar en las transacciones los puntos decisivos independientes y dependientes del sistema.
- Niveles de aislamiento para equilibrar la interferencia potencial con el trabajo adicional correspondiente en el control de concurrencia.
- Potencial para la pérdida de datos al utilizar el nivel de aislamiento READ COMMITTED.
- Especificación del tiempo de las restricciones para la puesta en práctica diferida de las restricciones de integridad hasta el final de una transacción.
- Puntos seguros para el regreso parcial al estado inicial de una transacción.
- Administración del flujo de trabajo para soportar el trabajo en colaboración.

## Preguntas

1. ¿Qué significa que una transacción sea un concepto definido por el usuario? ¿Por qué es importante que las transacciones sean definidas por el usuario?
2. Mencione las transacciones con las que interactuó durante la última semana.
3. Explique el propósito de los enunciados SQL START TRANSACTION, COMMIT y ROLLBACK. ¿Cómo varían en cada DBMS?
4. Explique brevemente el significado de las propiedades ACID. ¿De qué manera el control de concurrencia y la administración de recuperaciones soportan las propiedades ACID?
5. Explique brevemente el significado de transparencia en relación con el procesamiento de cómputo. ¿Por qué la transparencia es importante para el control de concurrencia y la administración de recuperaciones?
6. ¿Cuáles son los costos asociados con el control de concurrencia y la administración de recuperaciones? ¿Qué función debe desempeñar para evaluar estos costos: administrador de base de datos o programador de base de datos?
7. ¿Cuál es el objetivo del control de concurrencia? ¿Cómo es la medida que se utiliza con el objetivo en relación con el tiempo de espera?
8. ¿Qué es un punto decisivo? ¿Cómo se relacionan los puntos decisivos con los problemas de interferencia?
9. Analice las consecuencias de cada uno de los tipos de problemas de interferencia. ¿Cuál de ellos parece más serio?
10. ¿Qué es un candado? Explique en forma breve las diferencias entre candados compartidos (S) y exclusivos (X).
11. ¿Qué operaciones realiza el administrador de candados?
12. ¿Qué es un punto muerto y cómo se maneja?
13. ¿Qué es granularidad de los candados? ¿Cuáles son los intercambios al tener candados en un nivel de granularidad más fino en comparación con un nivel más alto?
14. ¿Qué es un candado de intención? ¿Por qué se utilizan en elementos con una granularidad más alta?
15. ¿Por qué la tercera condición de los candados 2PL casi siempre se simplifica, de modo que se liberan al final de una transacción?
16. ¿Cuál es el atractivo de los planteamientos optimistas de control de concurrencia? ¿Por qué no deben utilizarse aun cuando ofrecen un mejor desempeño?
17. Explique la diferencia entre almacenamiento volátil y no volátil.
18. Explique los efectos de las fallas locales, de sistema y de dispositivo en las transacciones activas y pasadas.
19. ¿Por qué la escritura forzada es la herramienta fundamental de la administración de recuperaciones?
20. ¿Qué tipo de datos redundantes se guardan en un registro? ¿Por qué la administración del registro es crítica para la recuperación?
21. ¿Qué es intervalo de control? ¿Cuál es el objetivo de determinar un intervalo de control?

22. ¿Qué procesamiento tiene lugar cuando ocurre un control consistente con el caché?
23. ¿Qué es un control confuso? ¿Cuáles son las ventajas de un control confuso en comparación con un control consistente con el caché?
24. ¿Qué es un control incremental? ¿De qué manera puede controlarse la cantidad de trabajo de reinicio con controles incrementales?
25. ¿Qué trabajo de reinicio es necesario para una falla de medios?
26. ¿Qué trabajo de reinicio es necesario para las fallas locales y de sistema en el planteamiento de actualización inmediata?
27. ¿Qué trabajo de reinicio es necesario para las fallas locales y de sistema en el planteamiento de actualización diferida?
28. ¿Qué es un límite de transacción? ¿Por qué la elección inapropiada del límite de transacción puede dar lugar a un mal desempeño?
29. ¿Qué criterios deben emplearse al seleccionar un límite de transacción?
30. ¿Por qué restricciones como las de débito-crédito deben ponerse en práctica como parte de una transacción en lugar de entre transacciones?
31. Explique la diferencia entre los puntos decisivos independientes del sistema y los dependientes del sistema. ¿Por qué es útil identificar los puntos decisivos?
32. Explique las tres elecciones para el límite de transacción de un formulario jerárquico.
33. ¿De qué manera un punto muerto puede estar influido por la elección del límite de transacción?
34. ¿Qué efecto secundario puede ocurrir al mover la interacción del usuario fuera del límite de la transacción?
35. ¿Cuál es el propósito de los niveles de aislamiento en SQL?
36. ¿De qué manera los niveles de aislamiento logran un acceso más concurrente?
37. ¿Qué nivel de aislamiento puede ser peligroso y por qué?
38. Mencione un ejemplo de una restricción para la cual es apropiada la actualización diferida.
39. ¿Qué enunciados y cláusulas de SQL comprenden la especificación del tiempo de las restricciones?
40. ¿Qué función desempeña un DBA al especificar el tiempo de las restricciones?
41. ¿Qué función desempeña un programador de bases de datos al especificar el tiempo de las restricciones?
42. ¿Cuál es el propósito de un punto guardado?
43. ¿Cómo se puede usar un punto guardado en la resolución de un punto muerto?
44. ¿Qué es un flujo de trabajo y cuál es su relación con las transacciones de bases de datos?
45. ¿Qué diferencias existen entre los flujos de trabajo orientados a los humanos y los orientados a las computadoras?
46. Mencione un ejemplo de un flujo de trabajo con tareas de alta complejidad y otro ejemplo con tareas muy estructuradas.
47. Analice las tecnologías que permiten la administración del flujo de trabajo. ¿Qué función desempeña la administración de transacciones en la administración del flujo de trabajo?
48. ¿Cuáles son las limitaciones de la administración de transacciones para soportar los flujos de trabajo?
49. ¿Qué relación existe entre los puntos de revisión incrementales y los procesos de recuperación?
50. ¿Qué nivel de participación es necesario para utilizar los servicios de control de concurrencia y recuperación que ofrece un DBMS?

## Problemas

Los problemas proporcionan práctica en el uso de los enunciados SQL que definen las transacciones, ponen a prueba sus conocimientos sobre el control de concurrencia y la administración de recuperaciones, y analizan las decisiones de diseño acerca de los límites de transacciones y los puntos decisivos.

1. Identifique dos transacciones que haya realizado recientemente. Defina el pseudocódigo para las transacciones al estilo de las figuras 15.1, 15.2 y 15.3.
2. Identifique los puntos decisivos en las transacciones del problema 1.
3. Con el uso de un cronograma, ilustre un problema de actualización perdida, utilizando las transacciones del problema 1 en caso de no usar el control de concurrencia.
4. Con el uso de un cronograma, ilustre un problema de dependencia sin realizar, utilizando las transacciones del problema 1 en caso de no usar el control de concurrencia.

5. Con el uso de un cronograma, ilustre un problema de lectura que no se pueda repetir, utilizando las transacciones del problema 1 en caso de no usar el control de concurrencia.
6. Con el uso de las transacciones del problema 1, explique si un punto muerto sería un problema en caso de usar un candado. En caso de que sea posible, utilice un cronograma para demostrar la existencia de un punto muerto con sus transacciones.
7. Utilice las siguientes tablas de la Base de Datos de Contabilidad y el Registro de Contabilidad para responder los problemas 7.1 a 7.7. Los comentarios se presentan después de las tablas y el formulario.

Account(AcctNo, Name, Address, Balance, LastCheckNo, StartDate)

Entry(EntryNo, AcctNo, Date, Amount, Desc)

Category(CatNo, Name, Description)

EntryLine(EntryNo, CatNo, Amount, Description)

Registro de contabilidad para la línea de crédito de Wells Fargo			
Número de entrada	E101	Fecha:	3/11/2006
Descripción:	Compras en OfficeMax	Cantidad:	\$442.00
Número de factura	I101		
Categoría	Descripción	Cantidad	
Artículos para oficina	Sobres	25.00	
Equipo	Máquina de fax	167.00	
Software para computadora	Actualización de MS Office	250.00	

- Las llaves primarias están subrayadas en las tablas. Las llaves foráneas están en cursivas.
  - El Registro de Contabilidad graba las actividades en una cuenta, como una línea de crédito o las cuentas por cobrar. El Registro de Contabilidad está diseñado para que lo use el departamento de contabilidad de empresas de tamaño moderado. El formulario muestra una entrada registrada, pero un registro contiene todas las entradas registradas desde que se abrió la cuenta.
  - El formulario principal se utiliza para insertar un registro en la tabla *Entry* y actualizar el campo *Balance* en la tabla *Account*. Las cuentas tienen un nombre único (Línea de Crédito de Wells Fargo) que aparece en el título del registro. Las cuentas tienen otros atributos que no aparecen en el formulario: un número único (el nombre también lo es), fecha de inicio, dirección, tipo (Cuenta por Cobrar, Inversión, Crédito, etc.) y saldo actual.
  - En el subformulario el usuario distribuye la cantidad total de la entrada en categorías. El campo *Category* es un cuadro combinado. Cuando el usuario hace clic en el campo de categoría, aparecen el número y el nombre de la categoría. La captura de una nueva línea en el subformulario inserta una fila en la tabla *EntryLine*.
  - El campo *Description* en el subformulario describe una fila en la tabla *EntryLine*, en lugar de la tabla *Category*.
- 7.1 ¿Cuáles son los límites de transacciones posibles para el formulario Registro de Contabilidad?
- 7.2 Seleccione un límite de transacción entre las elecciones del problema 7.1. Justifique su elección utilizando los criterios que se definen en la sección 15.4.1.
- 7.3 Identifique los puntos decisivos independientes del sistema que resultan del uso concurrente del Registro de Contabilidad (digamos, varios dependientes en el departamento de contabilidad). Explique por qué es decisivo cada uno de estos puntos.
- 7.4 Identifique los puntos decisivos dependientes del sistema que resultan del uso concurrente del Registro de Contabilidad (digamos, varios dependientes en el departamento de contabilidad). Puede suponer que el DBMS no puede utilizar candados más finos que una página de base de datos.
- 7.5 Describa un problema de actualización perdida que comprenda uno de los puntos de decisiones que podrían ocurrir con el uso concurrente del Registro de Contabilidad. Utilice un cronograma para ilustrar su ejemplo.
- 7.6 Describa una situación de lectura sucia que comprenda uno de los puntos decisivos que podrían ocurrir con el uso concurrente del Registro de Contabilidad. Utilice un cronograma para ilustrar su ejemplo.

- 7.7 ¿Es probable que un punto muerto sea un problema con el uso concurrente del Registro de Contabilidad? Considere el caso en el que los candados se mantienen hasta que se completan todas las líneas del subformulario. Explique por qué sí o por qué no. Si es probable un punto muerto, mencione un ejemplo como justificación. ¿El problema del punto muerto seguiría existiendo si los candados se conservaran sólo hasta terminar cada línea en el subformulario? Explique por qué sí o por qué no.
8. Utilice las siguientes tablas Patient y Patient Billing Form para responder los problemas 8.1 a 8.4. Los comentarios se presentan después de las tablas y el formulario.

**Patient**(PatSSN, PatName, PatCity, PatAge)  
**Doctor**(DocNo, DocName, DocSpecialty)  
**Bill**(BillNo, PatSSN, BillDate, AdmitDate, DischargeDate)  
**Charge**(ChgNo, BillNo, ItemNo, ChgDate, ChgQty, DocNo)  
**Item**(Itemno, ItemDesc, ItemUnit, ItemRate, ItemQOH)

- El formulario principal se utiliza para insertar un registro en la tabla *Bill*. Los campos de la tabla *Patient* son de sólo lectura en el formulario principal.
  - El subformulario se puede usar para insertar una nueva fila en la tabla *Charge*. Los campos de las tablas *Item* y *Doctor* son de sólo lectura.
  - Al capturar una línea en el subformulario se actualiza la fila del artículo asociado. El campo *Qty* del formulario afecta el valor actual en el campo *ItemQOH* (item quantity on hand: cantidad del artículo a la mano).
- 8.1 ¿Cuáles son los límites de transacciones posibles para el formulario Patient Billing?
- 8.2 Seleccione un límite de transacción entre las elecciones del problema 8.1. Justifique su elección utilizando los criterios definidos en la sección 15.4.1.
- 8.3 Identifique los puntos decisivos independientes del sistema que resultan del uso concurrente del formulario Patient Billing (digamos, varios prestadores de servicios de salud). Explique por qué es decisivo cada uno de estos puntos.
- 8.4 Identifique los puntos decisivos dependientes del sistema, que resultan del uso concurrente del formulario Patient Billing. Puede suponer que el DBMS no puede usar candados más finos que una página de la base de datos.
9. Utilice las siguientes tablas de la base de datos Airline Reservation y el Formulario Flight Reservation para responder los problemas 9.1 a 9.4. Los comentarios se presentan después de las tablas y el formulario.

**Flight**(FlightNo, DepCityCode, ArrCityCode, DepTime, ArrTime, FlgDays)  
**FlightDate**(FlightNo, FlightDate, RemSeats)  
**Reservation**(ResNo, CustNo, ResDate, Amount, CrCardNo)

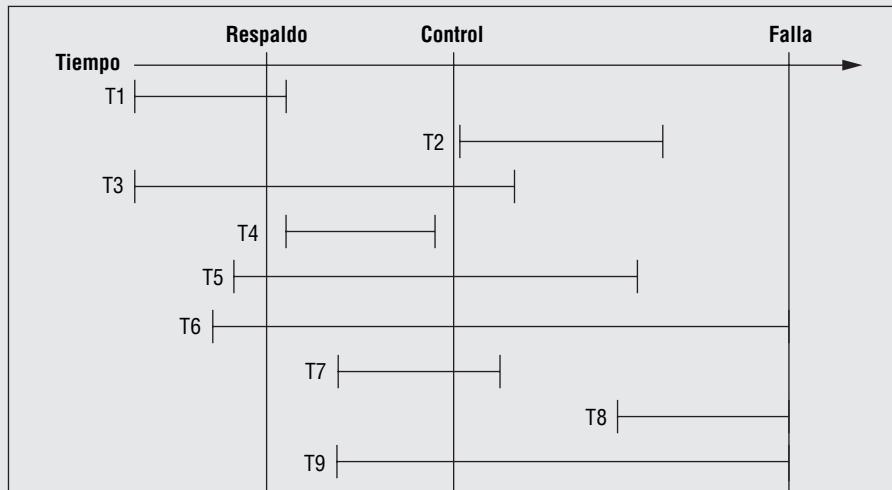
**ReserveFlight(ResNo, FlightNo, FlightDate)**

**Customer(CustNo, CustName, CustStreet, CustCity, CustState, CustZip)**

**City(CityCode, CityName, Altitude, AirportConditions)**

Formulario de reservación de vuelo					
Número de reserva:	R101	Fecha de hoy:	26/08/2006		
Número de tarjeta de crédito:	CC101	Cantidad:	\$442.00		
Número de cliente:	C101	Nombre del cliente:	Jill Horn		
Plan de vuelo					
Número de vuelo	Fecha	Ciudad de origen	Hora de salida	Ciudad de salida	Hora de llegada
F101	26/08/2006	DNV	10:30AM	CHG	11:45AM
F201	31/08/2006	CHG	10:00AM	DNV	1:20PM

- Las llaves primarias en las tablas están subrayadas. Las llaves foráneas están en cursivas. Observe que la combinación de *ResNo*, *FlightNo* y *FlightDate* es la llave primaria de la tabla *ReserveFlight*. La combinación de *FlightNo* y *FlightDate* es una llave foránea en la tabla *ReserveFlight*, que se refiere a la tabla *FlightDate*.
  - El Formulario de reservación de vuelo está simplificado, ya que sólo incluye una clase de asientos; no hay asientos reservados ni comidas. Sin embargo, las líneas de bajo costo tienen a menudo estas restricciones.
  - El formulario principal se utiliza para insertar un registro en la tabla *Reservation*. Los campos de la tabla *Customer* son de sólo lectura.
  - El subformulario se utiliza para insertar filas nuevas en la tabla *ReserveFlight* y actualizar el campo *RemSeats* en la tabla *FlightDate*. Los campos de la tabla *Flight* son de sólo lectura.
- 9.1 Seleccione un límite de transacción para el Formulario de reservación de vuelo. Justifique su elección utilizando los criterios definidos en la sección 15.4.1.
  - 9.2 Identifique los puntos decisivos independientes del sistema que resultan del uso concurrente del Formulario de reservación de vuelo (digamos, varios agentes de reservaciones). Explique por qué es decisivo cada uno de estos puntos.
  - 9.3 Identifique los puntos decisivos dependientes del sistema que resultan del uso concurrente del Formulario de reservación de vuelo. Puede suponer que el DBMS no puede usar candados más finos que una página de la base de datos.
  - 9.4 ¿Es probable que un punto muerto sea un problema con el uso concurrente del Formulario de reservación de vuelo? Si es así, mencione un ejemplo como justificación.



10. El siguiente cronograma muestra el estado de las transacciones en relación con el respaldo, el control y la falla más recientes. Utilice el cronograma para resolver los problemas en las subpartes de este problema (10.1 a 10.5).
  - 10.1 Describa el trabajo de reinicio si la transacción T3 se cancela (con un enunciado ROLLBACK) después del control pero antes de la falla. Suponga que el administrador de recuperaciones utiliza el planteamiento de actualización diferida.
  - 10.2 Describa el trabajo de reinicio si la transacción T3 se cancela (con un enunciado ROLLBACK) después del control pero antes de la falla. Suponga que el administrador de recuperaciones utiliza el planteamiento de actualización inmediata.
  - 10.3 Describa el trabajo de reinicio en caso de que ocurra una falla en el sistema. Suponga que el administrador de recuperaciones emplea el planteamiento de actualización diferida.
  - 10.4 Describa el trabajo de reinicio en caso de que ocurra una falla en el sistema. Suponga que el administrador de recuperaciones emplea el planteamiento de actualización inmediata.
  - 10.5 Describa el trabajo de reinicio si ocurre una falla en el dispositivo.
11. Utilice la World Wide Web para revisar las evaluaciones comparativas del procesamiento de transacciones. ¿Por qué la evaluación comparativa de débito-crédito ha sido reemplazada por otras? ¿Cuántas transacciones por minuto se reportan para los distintos DBMS? Revise el código para una o más transacciones comparativas. ¿Identifica algunos puntos decisivos en las transacciones?
12. Rediseñe la transacción en cajero automático (figura 15.1) para eliminar la interacción del usuario. Por favor, comente cualquier efecto secundario que pueda resultar de esta eliminación.
13. Rediseñe la transacción de pedido de productos (figura 15.3) para eliminar la interacción del usuario. Por favor, comente cualquier efecto secundario que pueda resultar de esta eliminación.
14. ¿Por qué algunos DBMS empresariales utilizan READ COMMITTED como el nivel de aislamiento predeterminado? Razone sobre las ventajas y desventajas de usar este nivel como nivel de aislamiento predeterminado. En su análisis, debe pensar con detenimiento sobre la importancia del problema de la actualización perdida.
15. Utilizando el siguiente registro de transacciones, cree una tabla que incluya las operaciones de registro para el planteamiento de actualización inmediata. Use la tabla 15.10 como formato para su respuesta.

LSN	TransNo	Acción	Hora	Tabla	Fila	Columna	Antiguo	Nuevo
1	1	START	2:09:20					
2	1	INSERT	2:09:21	Resv	1001	*		<101, 400, ... >
3	2	START	2:09:22					
4	1	UPDATE	2:09:23	Flight	2521	SeatsRem	10	9
5	2	INSERT	2:09:24	Resv	1107	*		<101, 400, ... >
6	2	UPDATE	2:09:25	Flight	3533	SeatsRem	3	2
7	3	START	2:09:26					
8	1	INSERT	2:09:27	Resv	1322	*		<102, 225, ... >
9	1	UPDATE	2:09:28	Flight	4544	SeatsRem	15	14
10		CKPT(1,2,3)	2:09:29					
11	2	INSERT	2:09:30	Resv	1255	*		<111, 500, ... >
12	2	UPDATE	2:09:31	Flight	3288	SeatsRem	2	1
13	1	COMMIT	2:09:32					
14	3	INSERT	2:09:33	Resv	1506	*		<151, 159, ... >
15	3	UPDATE	2:09:34	Flight	3099	SeatsRem	50	49
16	4	START	2:09:36					
17	3	INSERT	2:09:37	Resv	1299	*		<222, 384, ... >
18	3	UPDATE	2:09:38	Flight	4522	SeatsRem	25	24
19	4	INSERT	2:09:39	Resv	1022	*		<222, 384, ... >
20		CKPT(2,3,4)	2:09:40					
21	2	COMMIT	2:09:41					
22	4	UPDATE	2:09:42	Flight	2785	SeatsRem	1	0
23	3	COMMIT	2:09:43					
24	4	INSERT	2:09:44	Resv	1098	*		<515, 99, ... >
25	4	UPDATE	2:09:45	Flight	3843	SeatsRem	15	14

16. Utilizando el registro de transacciones del problema 15, cree una tabla que incluya las operaciones de registro para el planteamiento de actualización diferida. Use la tabla 15.11 como formato para su respuesta.
17. Identifique el problema de control de concurrencia ilustrado en el siguiente cronograma. Identifique el nivel de aislamiento menos restrictivo que elimine el problema. Observe que SERIALIZABLE es el nivel de aislamiento más restrictivo. Vuelva a trazar el cronograma para mostrar los candados que impone el nivel de aislamiento menos restrictivo que elimina el problema.

Transacción A	Tiempo	Transacción B
	$T_1$	UPDATE QOH <sub>2</sub> = QOH <sub>2</sub> - 5 (20)
Leer QOH <sub>2</sub> (20)	$T_2$	
Sum = Sum + QOH <sub>2</sub>	$T_3$	
Leer QOH <sub>1</sub> (15)	$T_4$	
Sum = Sum + QOH <sub>1</sub>	$T_5$	
	$T_6$	UPDATE QOH <sub>1</sub> = QOH <sub>1</sub> - 5 (13)
	$T_7$	Commit

18. Identifique el problema de control de concurrencia ilustrado en el siguiente cronograma. Identifique el nivel de aislamiento menos restrictivo que elimine el problema. Observe que SERIALIZABLE es el nivel de aislamiento más restrictivo. Vuelva a trazar el cronograma para mostrar los candados que impone el nivel de aislamiento menos restrictivo que elimina el problema.

Transacción A	Tiempo	Transacción B
	$T_1$	Leer QOH <sub>1</sub> (55)
	$T_2$	QOH <sub>1</sub> = QOH <sub>1</sub> - 10
	$T_3$	Escribir QOH <sub>1</sub> (45)
Leer QOH <sub>1</sub> (45)	$T_4$	
Leer QOH <sub>2</sub> (15)	$T_5$	
	$T_6$	Leer QOH <sub>2</sub> (15)
	$T_7$	QOH <sub>2</sub> = QOH <sub>2</sub> - 5
	$T_8$	Escribir QOH <sub>2</sub> (10)
	$T_9$	Rollback

19. Identifique el problema de control de concurrencia ilustrado en el siguiente cronograma. Identifique el nivel de aislamiento menos restrictivo que elimine el problema. Observe que SERIALIZABLE es el nivel de aislamiento más restrictivo. Vuelva a trazar el cronograma para mostrar los candados que impone el nivel de aislamiento menos restrictivo que elimina el problema.

Transacción A	Tiempo	Transacción B
	$T_1$	Leer QOH <sub>1</sub> (10)
	$T_2$	If QOH <sub>1</sub> > 10 entonces QOH <sub>1</sub> = QOH <sub>1</sub> + 30
Leer QOH <sub>1</sub> (10)	$T_3$	
QOH <sub>1</sub> = QOH <sub>1</sub> - 3	$T_4$	
	$T_5$	Escribir QOH <sub>1</sub> (40)
	$T_6$	Commit
Escribir QOH <sub>1</sub> (7)	$T_7$	

20. Identifique el problema de control de concurrencia ilustrado en el siguiente cronograma. Identifique el nivel de aislamiento menos restrictivo que elimine el problema. Observe que SERIALIZABLE es el nivel de aislamiento más restrictivo. Vuelva a trazar el cronograma para mostrar los candados que impone el nivel de aislamiento menos restrictivo que elimina el problema.

Transacción A	Tiempo	Transacción B
Leer QOH (10)	$T_1$	
$QOH = QOH + 30$	$T_2$	
	$T_3$	Leer QOH (10)
	$T_4$	$QOH = QOH - 10$
Escribir SR (40)	$T_5$	
	$T_6$	Escribir SR (0)
Commit	$T_7$	
	$T_8$	Commit

## Referencias para ampliar su estudio

Aunque este capítulo ofrece una cobertura amplia de la administración de transacciones, sólo cubre sus aspectos básicos. La administración de transacciones es un tema detallado sobre el que se han escrito libros completos. Algunos libros especializados en este tema son Bernstein y Newcomer (1997) así como Gray y Reuter (1993). Shasha y Bonnet (2003) ofrecen más detalles sobre el diseño de transacciones y puesta a punto de las recuperaciones. Peinl, Reuter y Sammer (1988) ofrecen un estudio de caso sobre el diseño de transacciones que explica con detalle las ideas presentadas en la sección 15.4. Para más detalles sobre el desempeño del procesamiento de transacciones, consulte la página principal del Transaction Processing Performance Council ([www.tpc.org](http://www.tpc.org)). Los sitios DBAZine ([www.dbazine.com](http://www.dbazine.com)) y DevX Database Zone ([www.devx.com](http://www.devx.com)) ofrecen consejos prácticos sobre el procesamiento de transacciones.

## Apéndice 15.A

### Resumen de sintaxis de SQL:2003

Este apéndice resume la sintaxis de SQL:2003 para la cláusula de tiempo de las restricciones, el enunciado SET CONSTRAINTS, el enunciado SET TRANSACTION y los enunciados de punto guardado que estudiamos en el capítulo. Las convenciones utilizadas en la anotación de la sintaxis son idénticas a aquellas que se usan al final del capítulo 3.

### Cláusula de tiempo para las restricciones

```

CREATE TABLE TableName
  ( <Column-Definition>* [ , <Table-Constraint>* ] )

<Column-Definition>: ColumnName DataType
  [ DEFAULT { DefaultValue | USER | NULL } ]
  [ <Column-Constraint> ]

<Column-Constraint>: [ CONSTRAINT ConstraintName ]
  { NOT NULL |
    <Foreign-Key-Constraint> | -- defined in Chapter 3
    <Uniqueness-Constraint> | -- defined in Chapter 3
    <Check-Constraint> } -- defined in Chapter 14
  [ <Timing-Clause> ]

```

```

<Table-Constraint>: [ CONSTRAINT ConstraintName ]
    { <Primary-Key-Constraint> | -- defined in Chapter 3
      <Foreign-Key-Constraint> | -- defined in Chapter 3
      <Uniqueness-Constraint> | -- defined in Chapter 3
      <Check-Constraint> } -- defined in Chapter 14
    [ <Timing-Clause> ]

<Timing-Clause>:
    { NOT DEFERRABLE |
      DEFERRABLE { INITIALLY IMMEDIATE |
                   INITIALLY DEFERRED } }

CREATE ASSERTION AssertionName
    CHECK ( <Group-Condition> ) [ <Timing-Clause> ]

<Group-Condition>: -- definido en el capítulo 4

```

## Enunciado SET CONSTRAINTS

```

SET CONSTRAINTS { ALL | ConstraintName* }
                { IMMEDIATE | DEFERRED }

```

## Enunciado SET TRANSACTION

```

SET [LOCAL] TRANSACTION <Mode>*
<Mode>: { <Isolation-Level> | <Access-Mode> |
           <Diagnostics> }

<Isolation-Level>: ISOLATION LEVEL
                    { SERIALIZABLE |
                      REPEATABLE READ |
                      READ COMMITTED |
                      READ UNCOMMITTED }

<Access-Mode>: { READ WRITE | READ ONLY }

<Diagnostics>: DIAGNOSTICS SIZE Constant

```

## Enunciados de punto guardado

```

SAVEPOINT <SavePointName> -- crea un punto guardado
RELEASE <SavePointName> -- elimina un punto guardado
ROLLBACK TO SAVEPOINT <SavePointName> -- regresa a un punto guardado

```

# Capítulo 16

## Tecnología y administración de data warehouse

### Objetivos de aprendizaje

Este capítulo ofrece los cimientos para una forma emergente de bases de datos, llamada data warehouse, que se usa cada vez más para apoyar la toma de decisiones. Después de este capítulo, el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Explicar las diferencias conceptuales entre las bases de datos operacionales y los data warehouse.
- Comprender arquitecturas para aplicar la tecnología de data warehouse en las organizaciones.
- Comprender la representación y manipulación de cubos de datos.
- Aplicar el modelado y la manipulación relacional de datos para datos multidimensionales.
- Explicar aspectos de la calidad de los datos y la función de las herramientas de extracción, transformación y carga para el mantenimiento de un data warehouse.
- Adquirir idea del complejo proceso de refrescar un data warehouse.

### Panorama general

Imagine al ejecutivo de un corporativo nacional minorista de electrónicos. “¿Qué tiendas minoristas fueron las principales productoras durante los últimos 12 meses en la región de Rocky Mountain?”. Las preguntas de seguimiento pueden incluir: “¿cuáles fueron los productos más rentables en las principales tiendas minoristas?”, y “¿cuáles fueron las más exitosas promociones de productos en las principales tiendas minoristas?”. Éstos son ejemplos de preguntas de apoyo a la toma de decisiones o de inteligencia de negocios que se hacen todos los días gerentes de todo el mundo. Las respuestas a estas preguntas a menudo requieren enunciados SQL que pueden tomar horas para codificarse y ejecutarse. Además, formular algunas de estas consultas puede requerir datos de un conjunto diverso de sistemas heredados internos y fuentes de mercado externas, implicando tanto bases de datos relacionales como no relacionales.

Preguntas para la toma de decisiones como las anteriores dan lugar a nuevos requerimientos de los DBMS. Este capítulo presenta la tecnología y administración de data warehouse para satisfacer los requerimientos del apoyo a la toma de decisiones. La tecnología del data warehouse

complementa y extiende la tecnología de base de datos relacionales más allá del procesamiento de transacciones en línea y las capacidades de consulta simple, como la cláusula GROUP BY en SQL. En este capítulo aprenderá inicialmente acerca de los requerimientos únicos para el procesamiento de un data warehouse, en contraposición con los requerimientos de procesamiento de transacciones que estudiamos en el capítulo 15. Después, aprenderá sobre el modelo multidimensional de datos y su implementación en bases de datos relacionales, con especial énfasis en las características de almacenamiento de datos en Oracle 10g. Aprenderá nuevas habilidades para el modelado y formulación de consultas que complementan el material de modelado de datos de la parte 3 y el material de formulación de consultas de las partes 2 y 5. Por último, aprenderá acerca del mantenimiento de un data warehouse, importante proceso que llevan a cabo los administradores de los data warehouse.

## 16.1 Conceptos básicos

---

El tipo de datos que se usa para fines de apoyo a la toma de decisiones es conceptualmente distinto al que se utiliza en las bases de datos de procesamiento de transacciones. Por lo tanto, también son diferentes las bases de datos que pueden almacenar dichos datos y las arquitecturas de cómputo que pueden procesarlos. Esta sección examina estas diferencias para proporcionar el fundamento para el estudio detallado de los aspectos tecnológicos y administrativos de secciones posteriores.

### 16.1.1 Procesamiento de transacciones versus apoyo a las decisiones

El procesamiento de transacciones implica diferentes necesidades en una organización que requiere apoyo a las decisiones. El procesamiento de transacciones, como se analizó en el capítulo 15, permite a las organizaciones hacer negocios diariamente de forma eficiente. Las bases de datos operacionales o de producción que se utilizan en el procesamiento de transacciones ayudan con decisiones como rastreo de pedidos, solución de quejas de los clientes y requerimientos de personal. Estas decisiones implican datos detallados acerca de los procesos empresariales.

En contraste, el procesamiento de apoyo a las decisiones ayuda a la alta gerencia a darle dirección a una organización a mediano y largo plazos. La gerencia necesita apoyo para toma de decisiones acerca de la capacidad de planeación, desarrollo de productos, localización de puntos de venta, promoción de productos y otras necesidades. Tradicionalmente, la mayoría de las organizaciones han supuesto que las bases de datos operacionales pueden proporcionar los datos para el apoyo a las decisiones. Conforme las organizaciones han desarrollado bases de datos operacionales para varias funciones, se ha desarrollado una brecha de información. Las organizaciones se han dado cuenta de modo gradual de que deben transformar significativamente las bases de datos para el apoyo a las decisiones.

Desde principios de la década de los noventa, se ha desarrollado un consenso sobre la necesidad de transformar las bases de datos operacionales para el apoyo a las decisiones. Las bases de datos operacionales pueden contener inconsistencias en áreas como formatos, identificación de entidades y unidades de medida que afectan su uso en el apoyo a las decisiones. Por otro lado, el apoyo a las decisiones necesita una visión amplia que integra procesos empresariales. Como resultado de los diferentes requerimientos, las bases de datos operacionales por lo general están separadas de las bases de datos para el apoyo a las decisiones. Utilizar una base de datos común para ambos tipos de procesamiento puede degradar en forma considerable el desempeño y hacer que sea difícil resumir la actividad en todos los procesos empresariales.

### 16.1.2 Características de los data warehouse

El data warehouse, término creado por William Inmon en 1990, se refiere a un depósito de datos central donde se integran, depuran y estandarizan los datos de bases de datos operacionales y otras fuentes para apoyar la toma de decisiones. Las actividades de transformación (depuración, integración y estandarización) son esenciales para lograr beneficios. Los beneficios tangibles de un data warehouse pueden incluir mayores ingresos y menores gastos permitidos por el análisis

**data warehouse**  
depósito central para datos resumidos e integrados de bases de datos operacionales y fuentes de datos externas.

empresarial, que no eran posibles antes de la utilización de los data warehouse. Por ejemplo, un data warehouse puede permitir menores pérdidas gracias a la mejor detección de fraudes, mejor retención de clientes a través del marketing dirigido y reducción en los costos implícitos de inventario por medio de un mejor pronóstico de la demanda.

Los requerimientos de procesamiento de las aplicaciones de apoyo a las decisiones han llevado a cuatro características distintivas para los data warehouse, mismas que son descritas a continuación:

1. **Orientado a los sujetos:** Un data warehouse se organiza en torno de los principales sujetos o entidades, como clientes, pedidos y productos. Esta orientación a los sujetos contrasta con la mayor orientación hacia los procesos en el procesamiento de transacciones.
2. **Integrado:** Los datos operativos de múltiples bases de datos y fuentes de datos externas se integran en un data warehouse para proporcionar una base de datos individual y unificada para el apoyo a las decisiones. La consolidación de los datos requiere convenciones de nomenclatura consistente, formatos de datos uniformes y escalas de medición comparables en las bases de datos y en las fuentes de datos externas.
3. **Variante de tiempo:** Los data warehouse usan marcas de tiempo para representar datos históricos. La dimensión del tiempo es crítica para identificar tendencias, pronosticar operaciones futuras y establecer objetivos de operación. Los data warehouse esencialmente consisten en una extensa serie de fotografías instantáneas, cada una de las cuales representa datos operativos capturados en un momento en el tiempo.
4. **No volátil:** Los datos nuevos en un data warehouse se anexan, en vez de reemplazarse, de modo que se preservan los datos históricos. El acto de anexar datos nuevos se conoce como refrescar el data warehouse. La falta de operaciones de actualización y eliminación garantiza que un data warehouse no contiene anomalías de actualización o eliminación. Los datos de las transacciones se transfieren a un data warehouse sólo cuando se ha completado la mayoría de las actividades de actualización.

La tabla 16.1 ilustra mejor las características de los data warehouse en comparación con las bases de datos operacionales. El procesamiento de transacciones depende de bases de datos operacionales con datos actuales en el plano individual, en tanto que el procesamiento de apoyo a las decisiones utiliza data warehouse con datos históricos tanto en el plano individual como en el resumido. Los datos del plano individual proporcionan flexibilidad para responder a una amplia variedad de necesidades de apoyo a las decisiones, mientras que los datos resumidos dan respuesta rápida a consultas repetitivas. Por ejemplo, una transacción de captura de pedidos requiere datos acerca de clientes individuales, pedidos y artículos en el inventario, en tanto que una aplicación de apoyo a las decisiones puede usar ventas mensuales a los clientes durante un periodo de varios años. Por consiguiente, las bases de datos operacionales tienen una orientación hacia los procesos (por ejemplo, todos los datos relevantes para un proceso empresarial particular), en comparación con una orientación hacia los sujetos de los data warehouse (por ejemplo, todos los datos de los clientes o todos los datos de los pedidos). Una transacción generalmente actualiza sólo unos cuantos registros, mientras que una aplicación de apoyo a las decisiones puede hacer consultas entre miles o millones de registros.

**TABLA 16.1**  
**Comparación de bases de datos operacionales y los data warehouse**

Característica	Base de datos operacional	Data warehouse
Actualidad	Actual	Histórico
Nivel de detalle	Individual	Individual y resumen
Orientación	Orientación hacia los procesos	Orientación hacia los sujetos
Número de registros procesados	Unos cuantos	Miles
Nivel de normalización	Normalizada en su mayoría	Violaciones frecuentes de la BCNF
Nivel de actualización	Volátil	No volátil (refrescado)
Modelo de datos	Relacional	Modelo relacional con esquemas de estrella y modelo multidimensional con cubos de datos

La integridad de los datos y el desempeño del procesamiento de transacciones requiere que las bases de datos operacionales estén muy normalizadas. En contraste, los data warehouse por lo regular son desnormalizados por la Forma Normal Boyce-Codd para reducir el esfuerzo de unir tablas grandes. La mayor parte del procesamiento de un data warehouse implica recuperaciones e inserciones periódicas de nuevos datos. Estas operaciones no sufren de anomalías causadas por un diseño que no esté totalmente normalizado.

Debido a los diferentes requerimientos de procesamiento, se han desarrollado distintos modelos de datos para las bases de datos operacionales y los data warehouse. El modelo de datos relacional predomina en el caso de las bases de datos operacionales. Durante los primeros años de uso del data warehouse dominaba el modelo multidimensional de datos. En años recientes, las bases de datos relacionales se han empleado para los data warehouse con un patrón de esquema conocido como esquema de estrella. El modelo de datos multidimensional se utiliza ahora como una representación para el usuario final de la vista de un data warehouse.

### 16.1.3 Arquitecturas para data warehouse

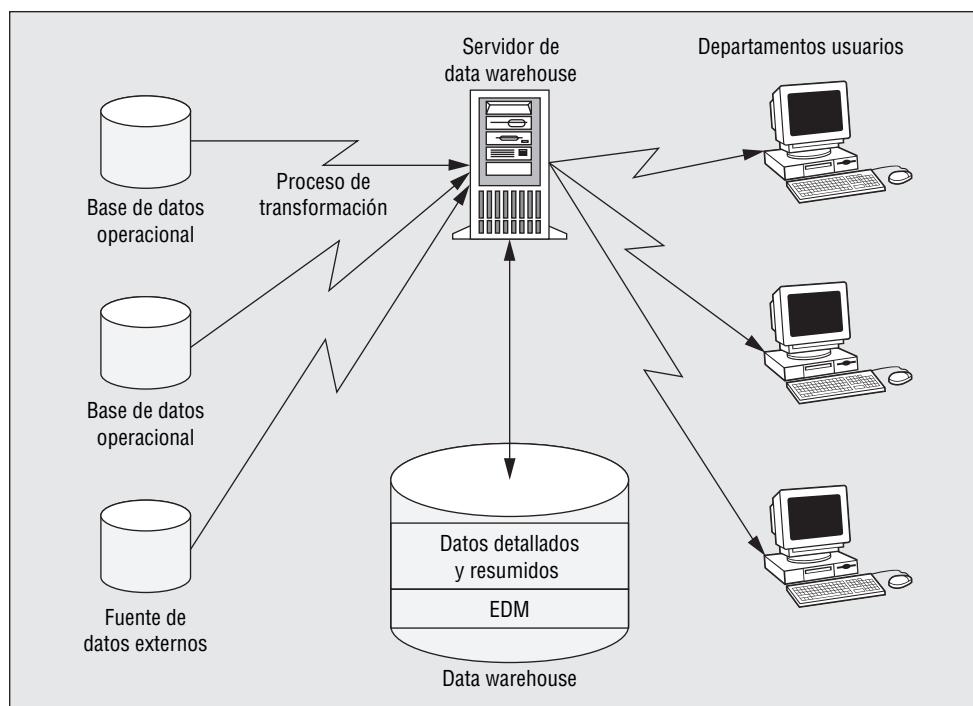
A pesar de los beneficios potenciales de un data warehouse, muchos proyectos de data warehouse han fracasado como resultado de una planeación deficiente. Los proyectos de data warehouse son grandes esfuerzos que implican coordinación entre muchas partes de una organización. Muchas organizaciones han subestimado el tiempo y el esfuerzo para conciliar diferentes vistas de un data warehouse. Además, el verdadero tamaño de un data warehouse puede llevarlo a un desempeño deficiente. Una arquitectura apropiada puede ayudar a aliviar problemas con el desempeño del data warehouse y el esfuerzo de desarrollo.

Para la mayoría de las organizaciones, es apropiada una arquitectura de data warehouse de dos o tres niveles. En una arquitectura de data warehouse de dos niveles (figura 16.1), los datos operativos se transforman y luego se transfieren a un data warehouse. Puede emplearse un plano separado de servidores para soportar las complejas actividades del proceso de transformación. Para asistir con el proceso de transformación, se crea un modelo de datos empresariales (EDM). El EDM describe la estructura del data warehouse y contiene los metadatos requeridos para entrar a bases de datos operacionales y fuentes de datos externas. El EDM también puede contener detalles acerca de la depuración e integración de las fuentes de datos. La gerencia usa el data warehouse directamente para recuperar datos para el apoyo a las decisiones.

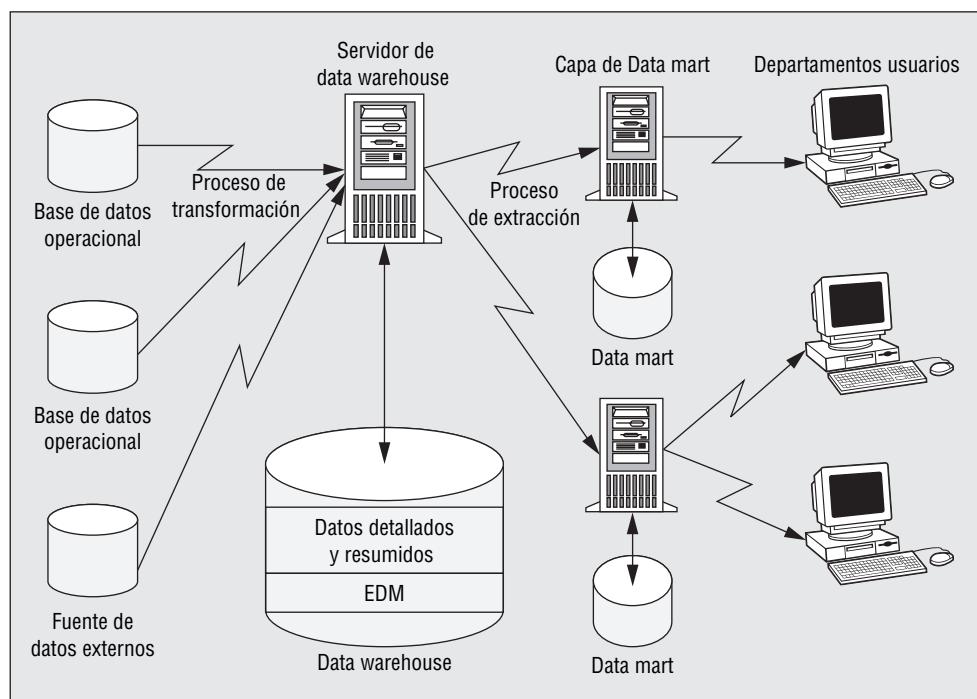
#### modelo de datos empresariales (EDM)

modelo de datos conceptual del data warehouse que define la estructura del data warehouse y los metadatos para tener acceso y transformar las bases de datos operacionales y las fuentes de datos externas.

**FIGURA 16.1**  
Arquitectura de data warehouse de dos niveles



**FIGURA 16.2**  
Arquitectura de data warehouse de tres niveles



La arquitectura de dos niveles puede tener problemas de desempeño para data warehouse grandes con aplicaciones intensivas de datos para el apoyo a las decisiones. Para resolver estas dificultades, muchas organizaciones grandes utilizan una arquitectura de tres niveles de data warehouse, como se muestra en la figura 16.2. Los usuarios departamentales generalmente necesitan acceso a pequeñas porciones del data warehouse y no al almacén entero. Para proporcionarles rápido acceso mientras se les aísla de los datos que necesitan otros grupos de usuarios, con frecuencia se usan data warehouse más pequeños llamados data marts. Los data marts actúan como la interfaz entre los usuarios finales y el data warehouse corporativo, almacenando un subconjunto del data warehouse y refrescándolo periódicamente (por ejemplo, diaria o semanalmente). Por lo regular, el data warehouse y los data marts residen en diferentes servidores para mejorar el desempeño y la tolerancia a errores. Los usuarios departamentales retienen el control de sus propios data marts, en tanto que el data warehouse permanece bajo control del personal de sistemas de información de la corporación.

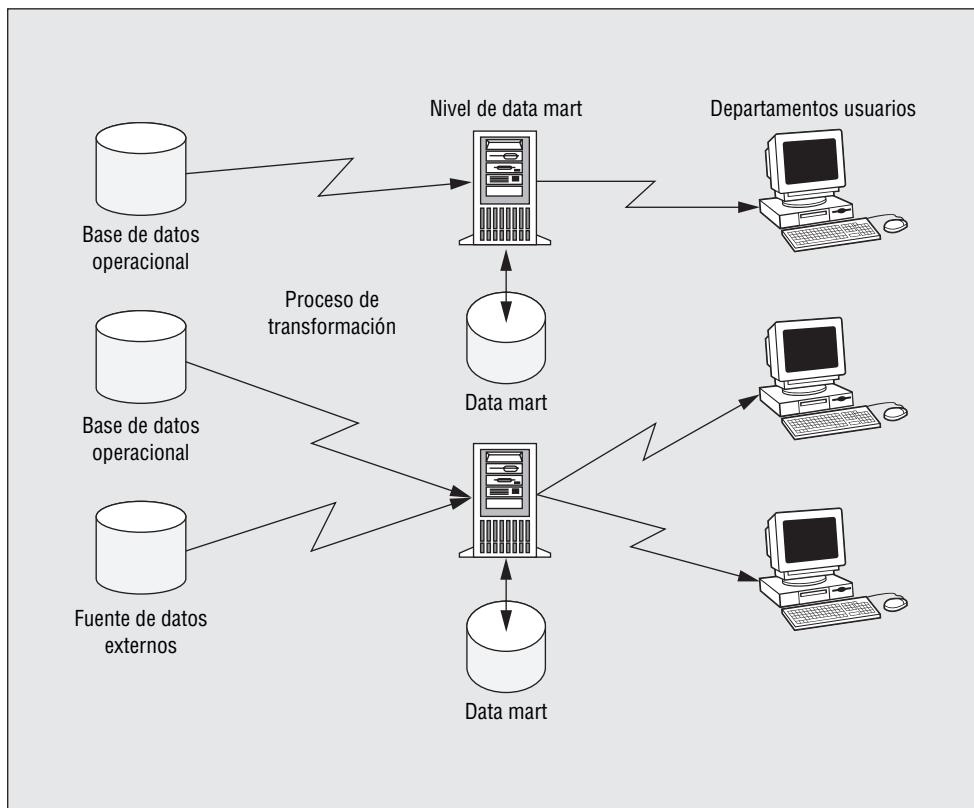
Dado el contexto de generalización en toda la empresa de un data warehouse, algunas organizaciones creen que la construcción de un data mart puede ser un retraso innecesario con oportunidades de negocios perdidas si se pone demasiado énfasis en crear primero un modelo de datos empresariales. En su lugar, algunas organizaciones emplean un planteamiento ascendente, como se ilustra en la figura 16.3. En una arquitectura ascendente de data warehouse, los datos se modelan una entidad a la vez y se les almacena en data marts separados. Con el paso del tiempo, se sintetizan, depuran y fusionan nuevos datos en data marts existentes o se les integra en nuevos data marts. El conjunto de data marts puede evolucionar a un data warehouse grande si la organización puede justificar la erogación de construir un modelo de datos empresariales.

Un desarrollo más reciente es el surgimiento de oper marts (abreviación de mercados operacionales –operational mart–), como lo señala Imhoff (2001). Un mercado operacional es un data mart justo-a-tiempo, normalmente construido a partir de una base de datos operacional en anticipación o en respuesta a eventos mayores, como desastres e introducciones de nuevos productos. Un mercado operacional da soporte a la demanda pico para los reportes y análisis empresarial que acompañan a un evento mayor. Después de que se atiende la demanda de apoyo a las decisiones, se puede desmantelar el mercado operativo o se puede fusionar en un data warehouse existente.

### data mart

subconjunto o vista de un data warehouse, normalmente en un departamento o nivel funcional, que contiene todos los datos requeridos para las tareas del apoyo a las decisiones de ese departamento.

**FIGURA 16.3**  
Arquitectura ascendente de data warehouse



#### 16.1.4 Minería de datos

**minería de datos**  
proceso de descubrimiento de patrones implícitos en datos almacenados en un data warehouse y el uso de dichos patrones para una ventaja empresarial.

Los data warehouse mejoran la calidad de la toma de decisiones al consolidar y agregar datos de transacciones. Es posible aumentar el valor de un data warehouse si se pueden descubrir patrones ocultos en los datos. La minería de datos se refiere al proceso de descubrimiento de patrones implícitos en datos almacenados en un data warehouse y la utilización de esos patrones como una ventaja empresarial. La minería de datos facilita la capacidad de detectar, entender y pronosticar patrones.

La aplicación más común de las técnicas de minería de datos es el marketing de objetivo. Las compañías de ventas por correo pueden incrementar los ingresos y disminuir los costos si pueden identificar a probables clientes y eliminar a los clientes con pocas probabilidades de compra. Las técnicas de minería de datos permiten a las personas que toman decisiones enfocarse en los esfuerzos de marketing por los datos demográficos y psicográficos de los clientes. Las industrias minorista, bancaria, de viajes y de bienes de consumo también han aprovechado las técnicas de minería de datos. Por ejemplo, la industria minorista usa técnicas de minería de datos para dirigir promociones y cambiar la combinación y orden de los artículos en las tiendas.

La minería de datos está considerada como un anexo de un data warehouse maduro. La minería de datos necesita datos más detallados que los que proporcionan los data warehouse tradicionales. Los volúmenes de datos y su dimensionalidad pueden ser mucho mayores en el caso de las técnicas de minería de datos que en los de las otras herramientas de análisis del data warehouse. Las técnicas de minería de datos se enriquecen con datos depurados de transacciones altamente dimensionales. Para soportar estos requerimientos de minería de datos, muchos data warehouse ahora almacenan datos detallados en el nivel del cliente individual, producto y otros.

La minería de datos requiere una serie de herramientas que se extienden más allá de las herramientas de análisis estadístico tradicional. Las herramientas de análisis estadístico tradicional no son adecuadas para datos muy dimensionales con una mezcla de datos numéricos y categóricos. Además, las técnicas estadísticas tradicionales no escalan bien a grandes cantidades de datos. La minería de datos incluye generalmente los siguientes tipos de herramientas:

- Herramientas de acceso a datos para extraer y muestrear datos de transacciones de acuerdo con criterios complejos para grandes bases de datos.
- Herramientas de visualización de datos que permiten que una persona que toma decisiones adquiera una comprensión más profunda e intuitiva de los datos.
- Una rica colección de modelos para agrupar, pronosticar y determinar reglas de asociación de grandes cantidades de datos. Los modelos implican redes neuronales, algoritmos genéticos, inducción en árboles de decisiones, algoritmos para descubrir reglas, redes de probabilidad y otras tecnologías de sistema experto.
- Una arquitectura que proporciona optimización, procesamiento cliente-servidor y consultas paralelas para escalar a grandes cantidades de datos.

Como complemento para el almacenamiento de datos, la minería de datos ofrece ideas que pueden eludir las técnicas tradicionales. La minería de datos sostiene la promesa de influir de manera más eficaz en los data warehouse al dar la capacidad para identificar relaciones ocultas en los datos que ahí se almacenan; facilita un descubrimiento guiado por los datos, usando técnicas como la creación de reglas de asociación (por ejemplo, entre el presupuesto para publicidad y las ventas de temporada), generación de perfiles (por ejemplo, patrones de compra para un segmento de clientes específico) y demás. Se puede emplear este conocimiento para mejorar las operaciones empresariales en áreas críticas, habilitar esfuerzos de marketing de objetivo y mejorar el servicio al cliente así como para la detección de fraudes.

### **16.1.5 Aplicaciones de los data warehouse**

Los proyectos de data warehouse se emprenden generalmente por razones competitivas: con el fin de lograr una ventaja estratégica o seguir siendo competitivos. Es frecuente que se emprenda un proyecto de data warehouse como parte de una estrategia corporativa para cambiar de un enfoque en el producto a un enfoque en el cliente. Los data warehouse exitosos han ayudado a identificar nuevos mercados, concentrar los recursos en clientes rentables, mejorar la retención de clientes y reducir los costos de inventario. Después del éxito de las organizaciones pioneras, otras organizaciones han seguido el ejemplo para permanecer en niveles competitivos.

Los proyectos de data warehouse se han emprendido en una gran variedad de industrias. Unas cuantas aplicaciones clave han llevado a la adopción de proyectos de data warehouse, como se enumera en la tabla 16.2. Las industrias altamente competitivas, como la venta al detalle, seguros, aerolíneas y telecomunicaciones (particularmente de servicio de larga distancia), han invertido con anticipación en tecnología y proyectos de data warehouse. Las industrias menos competitivas, como las de servicios públicos reguladas, han tardado más en invertir, a pesar de que incrementan las inversiones conforme maduran la tecnología y la práctica del data warehouse.

La madurez en la utilización de data warehouse varía entre industrias y organizaciones. Los primeros en adoptar data warehouse los han desplegado desde principios de la década de los noventa, mientras que quienes los adoptaron después lo hicieron hasta finales de la misma década. Con el rápido desarrollo de la tecnología de data warehouse y mejores prácticas, se necesita de una inversión continua en estos rubros con el objeto de sostener el valor de la empresa. Para ofrecer una guía acerca de las decisiones de inversión y mejores prácticas, las organizaciones se interesan en comparaciones con organizaciones semejantes para medir el nivel de uso del data warehouse.

**TABLA 16.2**  
**Aplicaciones del almacenamiento de datos por industria**

Industria	Aplicaciones clave
Aerolíneas	Administración de rendimiento, evaluación de rutas
Telecomunicaciones	Retención de clientes, diseño de redes
Seguros	Evaluación de riesgos, diseño de productos, detección de fraudes
Venta al detalle	Marketing de objetivo, administración de la cadena de abastecimiento

**TABLA 16.3**  
**Etapas del modelo de maduración del data warehouse**

Fuente: Eckerson 2004.

Etapa	Alcance	Arquitectura	Uso administrativo
Prenatal	Sistema operativo	Informes de administración	Centro de costos
Infantil	Analistas empresariales individuales	Hojas de cálculo	Discernimientos de administración
Niñez	Departamentos	Mercados de datos	Apoyo al análisis empresarial
Adolescencia	Divisiones	Data warehouse	Rastro de procesos empresariales
Edad adulta	Empresa	Data warehouse empresarial	Guía de la organización
Sabiduría	Inter–empresas	Servicios web y redes externas	Guía del mercado y la industria

El modelo de madurez del data warehouse se ha propuesto con el fin de ofrecer una guía para las decisiones de inversión en data warehouse (Eckerson 2004). El modelo de madurez consiste de seis etapas, las cuales se resumen en la tabla 16.3. Las etapas proporcionan un marco para observar el progreso de una organización, no una métrica absoluta, ya que las organizaciones pueden demostrar aspectos de múltiples etapas al mismo tiempo. Conforme las organizaciones pasan de etapas inferiores a más avanzadas, puede incrementarse el valor de la empresa. No obstante, a las organizaciones se les puede dificultar la justificación de nuevas inversiones importantes en data warehouse en las etapas de adolescencia y edad adulta, puesto que en ocasiones no es fácil cuantificar los beneficios.

Un aspecto importante del modelo de madurez es la dificultad de moverse entre ciertas etapas. En el caso de las organizaciones pequeñas pero en crecimiento, pasar de la infancia a la etapa de niñez puede ser difícil porque se necesita una inversión significativa en tecnología de data warehouse. Por lo que respecta a las grandes organizaciones, se lucha por pasar de la adolescencia a la edad adulta. Para hacer la transición, la alta gerencia debe percibir el data warehouse como un recurso empresarial vital, no sólo como una herramienta que proporciona el departamento de tecnología de la información.

## 16.2 Representación multidimensional de los datos

Después de entender los requerimientos únicos de datos para el apoyo a las decisiones, usted está listo para aprender acerca de la tecnología para satisfacer los requerimientos. El modelo multidimensional de datos soporta la representación y operación de datos especialmente diseñados para el procesamiento de apoyo a las decisiones en los data warehouse. Esta sección describe la terminología y operaciones del modelo multidimensional de datos.

### 16.2.1 Ejemplo de un cubo de datos multidimensional

Considere una compañía que vende productos electrónicos en distintas partes de Estados Unidos. En parte, la compañía comercializa cuatro diferentes productos de impresión (láser monocromática, inyección de tinta, para fotografías y portátil) en cinco estados diferentes (California, Washington, Colorado, Utah y Arizona). Para almacenar los datos de las ventas diarias para cada producto y cada ubicación en la base de datos relacional, necesita la tabla 16.4, la cual consta de tres columnas (*Product*, *Location* y *Sales*) y 20 filas (cuatro casos de *Product* por cinco casos de *Location*).

La representación de la tabla 16.4 puede ser compleja y pesada. Primero, imagine que la compañía desea agregar un quinto producto (digamos, láser a color). Con el fin de rastrear las ventas por estados de este nuevo producto, necesita agregar cinco filas, una por cada estado. En segundo lugar, debemos hacer notar que los datos de la tabla 16.4 representan datos de las ventas para un día en particular (por ejemplo, 10 de agosto de 2006). Para almacenar los mismos datos para el total de los 365 días del año 2006, necesita añadir una cuarta columna para almacenar la fecha de las ventas y duplicar las 20 filas para cada fecha 365 veces, con el objeto de

**TABLA 16.4**  
Representación relacional de los datos de ventas

Product	Location	Sales
Mono Laser	California	80
Mono Laser	Utah	40
Mono Laser	Arizona	70
Mono Laser	Washington	75
Mono Laser	Colorado	65
Ink Jet	California	110
Ink Jet	Utah	90
Ink Jet	Arizona	55
Ink Jet	Washington	85
Ink Jet	Colorado	45
Photo	California	60
Photo	Utah	50
Photo	Arizona	60
Photo	Washington	45
Photo	Colorado	85
Portable	California	25
Portable	Utah	30
Portable	Arizona	35
Portable	Washington	45
Portable	Colorado	60

**TABLA 16.5**  
Representación multidimensional de los datos de ventas

Location	Product			
	Mono Laser	Ink Jet	Photo	Portable
California	80	110	60	25
Utah	40	90	50	30
Arizona	70	55	60	35
Washington	75	85	45	45
Colorado	65	45	85	60

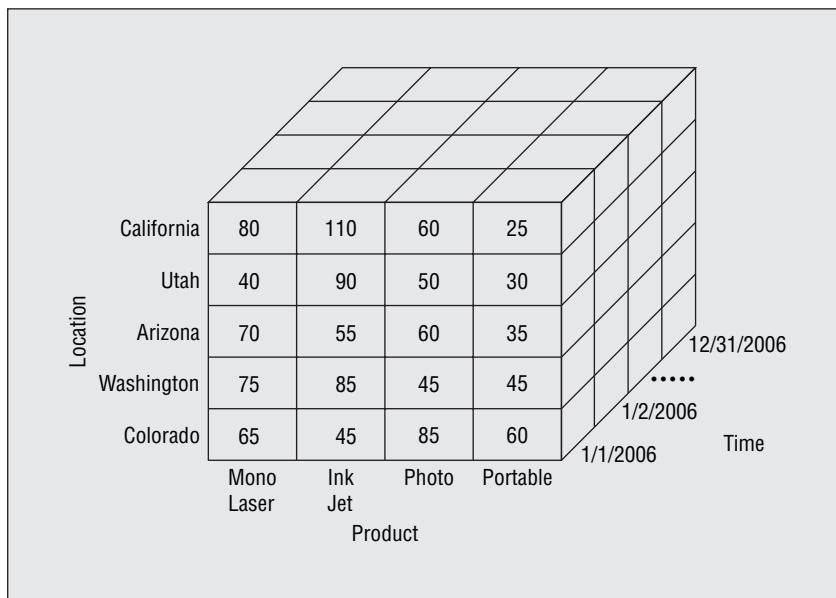
sumar una cuarta columna, para dar un total de 7300 filas. De igual modo, si quiere almacenar datos históricos para un periodo de 10 años, requiere 73 000 filas. Cada nueva fila debe contener los valores de producto, estado y fecha.

Un análisis de los datos de la tabla 16.4 revela que los datos contienen dos dimensiones, *Product* y *Location*, y un valor numérico para las ventas unitarias. Por lo tanto, es posible simplificar conceptualmente la tabla 16.4 al reordenar los datos en un formato multidimensional (véase tabla 16.5).

Es sencillo comprender y extender la representación multidimensional. Por ejemplo, sumar un quinto producto requiere una columna adicional a la derecha de la tabla 16.4. Para añadir fechas, es necesaria un tercera dimensión llamada *Time*, dando como resultado un arreglo tridimensional, como se muestra en la figura 16.4. Puede conceptualizar esta tabla tridimensional como un libro que consta de 365 páginas, cada página almacena datos de las ventas por producto y estado para una fecha específica del año. Además, la tabla multidimensional es más compacta porque las etiquetas de fila y columna no se duplican como en la tabla 16.3.

La representación multidimensional también ofrece una conveniente muestra de los totales sumatorios. Cada dimensión en un cubo puede contener totales que un usuario puede identificar con facilidad (totales de fila, totales de columna, totales de profundidad y totales generales). Por ejemplo, para añadir totales de fila a la tabla 16.5, puede agregarse una columna *Totals* con un valor por fila como se muestra en la tabla 16.6. En la representación relacional que se ilustra en la tabla 16.4, es preciso añadir totales utilizando valores nulos para los valores de columna. Por ejemplo, para representar el total de ventas de California para todos los productos, tiene que sumarse la fila <–, California, 275> a la tabla 16.4, donde el signo – indica todos los productos.

**FIGURA 16.4**  
Cubo de datos tridimensional



**TABLA 16.6**  
Representación multidimensional de los datos de ventas con totales de fila

Location	Product				Totals
	Mono Laser	Ink Jet	Photo	Portable	
California	80	110	60	25	275
Utah	40	90	50	30	210
Arizona	70	55	60	35	220
Washington	75	85	45	45	250
Colorado	65	45	85	60	255

Además de las ventajas de utilidad, la representación multidimensional puede proporcionar una mayor velocidad de recuperación. El almacenamiento directo de datos multidimensionales pone en evidencia la necesidad de convertir una representación de tabla en una representación multidimensional. No obstante, la representación multidimensional puede sufrir almacenamiento excesivo porque muchas celdas pueden permanecer vacías. Aun con técnicas de compresión, las grandes tablas multidimensionales pueden consumir considerablemente más espacio de almacenamiento que las tablas relacionales correspondientes.

En síntesis, una representación multidimensional proporciona una interfaz intuitiva para los analistas empresariales. Conforme aumenta el número de dimensiones, los analistas empresariales encuentran una representación multidimensional fácil de entender y visualizar en comparación con una representación relacional. Como la representación multidimensional satisface las necesidades de los analistas empresariales, ésta se emplea en gran medida en las herramientas de reportes empresariales, aun cuando las tablas relacionales ofrecen almacenamiento histórico.

**cubo de datos**  
Formato multidimensional en el que las celdas contienen datos numéricos, llamados medidas, organizados por temas, denominados dimensiones. En ocasiones, al cubo de datos se le conoce como hipercubo porque conceptualmente puede tener un número ilimitado de dimensiones.

## 16.2.2 Terminología multidimensional

Un cubo de datos generaliza las representaciones bidimensional (véase tabla 16.5) y tridimensional (véase figura 16.4) que estudiamos en la sección anterior. Un cubo de datos consiste en celdas que contienen medidas (valores numéricos, como las cantidades de ventas unitarias) y dimensiones para etiquetar o agrupar datos numéricos (por ejemplo, *Product*, *Location* y *Time*). Cada dimensión contiene valores que reciben el nombre de miembros. Por ejemplo, la dimensión *Location* tiene cinco miembros en la tabla 16.4 (California, Washington, Utah, Arizona y

Colorado). Tanto las dimensiones como las medidas pueden almacenarse o derivarse. Por ejemplo, la fecha de compra es una dimensión almacenada con el año, el mes y el día de las compras como dimensión derivada.

### *Detalles de dimensión*

Las dimensiones pueden tener jerarquías compuestas por niveles. Por ejemplo, la dimensión *Location* puede tener una jerarquía compuesta por los niveles país, estado y ciudad. De igual forma, la dimensión *Time* puede tener una jerarquía compuesta por año, trimestre, mes y fecha. Pueden usarse jerarquías para descender de los niveles superiores al detalle (por ejemplo, país) y trabajar en dirección opuesta. A pesar de que las jerarquías no son esenciales, permiten una representación conveniente y eficiente. Sin jerarquías, la dimensión *Location* debe contener el nivel más detallado (ciudad). Sin embargo, esta representación puede ser difícil para calcular totales en toda la dimensión. Alternativamente, es posible dividir la dimensión *Location* en dimensiones separadas para país, estado y ciudad, resultando en un cubo más grande.

Con fines de flexibilidad, las dimensiones pueden tener múltiples jerarquías. En una dimensión con múltiples jerarquías, por lo regular se comparte por lo menos un nivel. Por ejemplo, la dimensión *Location* puede tener una jerarquía con los niveles país, estado y ciudad y una segunda jerarquía con los niveles país, estado y código postal. La dimensión *Time* puede tener una jerarquía con los niveles año, trimestre y fecha, y una segunda jerarquía con los niveles año, semana y fecha. Las jerarquías múltiples permiten organizaciones alternativas para una dimensión.

Otra característica de la dimensión es la jerarquía irregular para una relación autorreferenciada entre miembros del mismo nivel. Por ejemplo, una dimensión de gerente podría tener una jerarquía irregular para mostrar las relaciones entre miembros y subordinados. Un analista que manipula un cubo de datos tal vez quiera expandir o contraer la dimensión de gerente de acuerdo con las relaciones entre gerentes y subordinados.

La selección de dimensiones influye en la dispersión de un cubo de datos. La dispersión indica el grado de celdas vacías en un cubo de datos. La dispersión puede ser un problema si se relacionan dos o más dimensiones. Por ejemplo, puede haber celdas vacías si ciertos productos se venden sólo en estados seleccionados. Si existe una gran cantidad de celdas vacías, el cubo de datos puede desperdiciar espacio y hacer que el proceso sea más lento. Es posible utilizar técnicas de compresión especial para reducir el diseño de los cubos de datos dispersos.

### *Detalles de medida*

Las celdas de un cubo de datos contienen medidas como los valores de ventas de la figura 16.4. Las medidas soportan operaciones numéricas como aritmética simple, cálculos estadísticos y ecuaciones simultáneas. Una celda puede contener una o más medidas. Por ejemplo, el número de unidades puede ser otra medida para el cubo de datos de ventas. El número de celdas ocupadas en un cubo multidimensional debe equivaler al número de filas en la tabla relacional correspondiente (la tabla 16.5 contiene 20 celdas ocupadas que corresponden a 20 filas de la tabla 16.4).

Las medidas derivadas pueden almacenarse en un cubo de datos o calcularse a partir de otras medidas en tiempo de operación. Las medidas que se pueden derivar de otras medidas en la misma celda normalmente no se almacenarían. Por ejemplo, el total de ventas en dólares puede calcularse como el total de ventas unitarias por las medidas de precio unitario en una celda. Las medidas de resumen derivadas de un serie de celdas pueden almacenarse o calcularse dependiendo del número de celdas y del costo de tener acceso a las celdas para el cálculo.

### *Otros ejemplos de cubo de datos*

Como indica esta sección, los cubos de datos pueden extenderse más allá del ejemplo de tres dimensiones que se muestra en la figura 16.4. La tabla 16.7 lista cubos de datos comunes para soportar la administración de recursos humanos y el análisis financiero. Las dimensiones con diagonales indican dimensiones jerárquicas. Las dimensiones de tiempo y ubicación también son jerárquicas, pero los posibles niveles no se listan porque pueden ser específicos para la organización.

**TABLA 16.7**

**Cubos de datos para soportar la administración de recursos humanos y el análisis financiero**

Cubo de datos	Dimensiones comunes	Medidas comunes
Análisis del cambio de personal	Compañía/línea de negocio/departamento, ubicación, rango salarial, clasificación del puesto, tiempo	Búsqueda de personal para contrataciones, transferencias, terminaciones y retiros
Utilización de empleados	Compañía/línea de negocio/departamento, ubicación, rango salarial, clasificación del puesto, tiempo	Horas completas equivalentes (FTE; <i>full time equivalent</i> ), horas FTE normales, horas FTE de tiempo extra
Análisis de activos	Tipo de activo, años en banda de servicio, tiempo, cuenta, compañía/línea de negocio/departamento, ubicación	Costo, valor neto en libros, valor mercantil
Análisis de fabricantes	Fabricantes, ubicación, cuenta, tiempo, unidad empresarial	Importe total de la facturación

### 16.2.3 Datos de serie de tiempo

El tiempo es una de las dimensiones más comunes en un data warehouse y es útil para capturar tendencias y hacer pronósticos. Una serie de tiempo proporciona almacenamiento de todos los datos históricos en una celda, en vez de especificar una dimensión de tiempo separada. La estructura de una medida se hace más compleja con una serie de tiempo, pero el número de dimensiones se reduce. Además, muchas funciones estadísticas pueden operar directamente en los datos de la serie de tiempo.

Una serie de tiempo es un tipo de datos de arreglo con varias propiedades especiales, como se mencionan a continuación. El arreglo soporta una serie de valores, uno para cada periodo. Los ejemplos de series de tiempo incluyen cantidades de ventas semanales, precios diarios de cierres de acciones y salarios anuales de los empleados. La siguiente lista muestra las propiedades típicas de una serie de tiempo:

- **Tipo de datos:** Esta propiedad denota la clase de dato almacenado en los puntos de datos. Normalmente, el tipo de datos es numérico, como números de punto flotante, números decimales fijos o enteros.
- **Fecha de inicio:** Esta propiedad expresa la fecha de inicio del primer punto de datos; por ejemplo, 1/1/2006.
- **Calendario:** Esta propiedad contiene el año calendario apropiado para la serie de tiempo; por ejemplo, el año fiscal 2006. Un conocimiento extenso de las reglas del calendario, como determinar años bisiestos y días festivos integrados en el calendario, reduce el esfuerzo en el desarrollo del data warehouse.
- **Periodicidad:** Esta propiedad especifica el intervalo entre los puntos de datos. La periodicidad puede ser diaria, semanal, mensual, trimestral, anual (calendario o años fiscales), por hora, intervalos de 15 minutos, periodicidad personalizada, entre otros.
- **Conversión:** Esta propiedad especifica la conversión de datos unitarios en datos agregados. Por ejemplo, agregar ventas diarias en las ventas semanales requiere total agregado, mientras que añadir precios de acciones diarios en los precios semanales implica una operación de promedio.

### 16.2.4 Operaciones del cubo de datos

Para los cubos de datos se han propuesto varias operaciones de apoyo a las decisiones. Esta sección analiza las operaciones que se emplean con mayor frecuencia. Aún está en desarrollo un conjunto estándar de operaciones de cubo de datos y no todas las herramientas de data warehouse soportan actualmente todas las operaciones.

#### *Rebanada*

El cubo de datos puede contener un gran número de dimensiones y los usuarios a menudo necesitan concentrarse en un subconjunto de dimensiones para lograr algún discernimiento. El

**FIGURA 16.5**  
Ejemplo de operación de rebanada

(Ubicación X rebanada de producto para la fecha = 1/1/2006)

Location	Product			
	Mono Laser	Ink Jet	Photo	Portable
California	80	110	60	25
Utah	40	90	50	30
Arizona	70	55	60	35
Washington	75	85	45	45
Colorado	65	45	85	60

**FIGURA 16.6**  
Ejemplo de operación de resumen de rebanada

Location	Time				Total Sales
	1/1/2006	1/2/2006	...	...	
California	400	670	...	...	16,250
Utah	340	190	...	...	11,107
Arizona	270	255	...	...	21,500
Washington	175	285	...	...	20,900
Colorado	165	245	...	...	21,336

**FIGURA 16.7**  
Ejemplo de operación de dato

Location	Utah	40	90	50	30
		Mono Laser	Ink Jet	Photo	Portable

operador de rebanada recupera un subconjunto de datos similar al operador de restricción del álgebra relacional. En una operación de rebanada, se establecen una o más dimensiones en valores específicos y se despliega el cubo de datos restante. Por ejemplo, la figura 16.5 muestra el cubo de datos resultante de la operación de rebanada en el cubo de datos de la figura 16.4, donde Time = 1/1/2006 junto con otras dos dimensiones (*Location* y *Product*).

Una variación del operador de rebanada permite que una persona que toma decisiones resuma a todos los miembros en vez de enfocarse sólo en uno. El operador de resumen de rebanada reemplaza una o más dimensiones con cálculos de resumen. El cálculo de resumen con frecuencia indica el valor total de todos los miembros o la tendencia central de la dimensión como promedio o valor mediano. Por ejemplo, la figura 16.6 muestra el resultado de la operación de resumen de rebanada donde la dimensión *Product* se sustituye con la suma de las ventas de todos los productos. Puede agregarse una nueva columna llamada *Total Sales* para almacenar ventas generales de los productos para el año entero.

### Dado

Como las dimensiones individuales pueden contener un gran número de miembros, los usuarios necesitan enfocarse en un subconjunto de ellos para lograr algún discernimiento. El operador de dado reemplaza una dimensión con un subconjunto de valores de la dimensión. Por ejemplo, la figura 16.6 presenta el resultado de una operación de dato para desplegar las ventas del Estado de Utah para el 1 de enero de 2006. Las operaciones de dato por lo regular se dan después de una operación de rebanada y regresa un subconjunto de los valores desplegados en la rebanada previa. Ayudan a concentrar la atención en una o más columnas de número de una rebanada.

### Descenso

Los usuarios con frecuencia quieren navegar entre los niveles de las dimensiones jerárquicas. El operador de descenso permite a los usuarios navegar de un nivel más general a un nivel más

específico. Por ejemplo, la figura 16.8 muestra una operación de descenso en el Estado de Utah de la dimensión *Location*. El signo más en Utah indica una operación de descenso.

### *Ascenso*

El ascenso es lo contrario del descenso. El *ascenso* implica moverse de un nivel específico a un nivel más general de una dimensión jerárquica. Por ejemplo, una persona que toma decisiones puede ascender datos de las ventas de un nivel diario a uno trimestral para las necesidades de reportes de fin de trimestre. En el ejemplo de las ventas de impresoras, la figura 16.5 presenta un ascenso del Estado de Utah de la figura 16.8.

### *Pivote*

El operador de pivote soporta la reorganización de las dimensiones del cubo de datos. Por ejemplo, en la figura 16.8, la posición de las dimensiones *Product* y *Location* se puede revertir de modo que *Product* aparezca en las filas y *Location* en las columnas. El operador de pivote hace posible que se presente un cubo de datos con un orden visual más atractivo.

El operador de pivote se usa con mayor frecuencia en cubos de datos con más de dos dimensiones, en los que aparecen múltiples dimensiones en el área de la fila y/o columna porque no es posible desplegar dos dimensiones de otra manera. Por ejemplo, para desplegar un cubo de datos con las dimensiones *Location*, *Product* y *Time*, puede desplegarse la dimensión *Time* en el área de la fila dentro de la dimensión *Location*. Una operación de pivote podría reordenar el cubo de datos de forma tal que la dimensión *Location* se despliegue dentro de la dimensión *Time*.

### *Resumen de operadores*

Para ayudarle a recordar las operaciones del cubo de datos, la tabla 16.8 resume el propósito de cada operador.

**FIGURA 16.8**  
Operación de descenso para el Estado de Utah en la figura 16.5

<b>Location</b>	<b>Product</b>			
	<b>Mono Laser</b>	<b>Ink Jet</b>	<b>Photo</b>	<b>Portable</b>
California + Utah	80	110	60	25
Salt Lake	20	20	10	15
Park City	5	30	10	5
Ogden	15	40	30	10
Arizona	70	55	60	35
Washington	75	85	45	45
Colorado	65	45	85	60

**TABLA 16.8** Resumen de los operadores del cubo de datos

<b>Operador</b>	<b>Propósito</b>	<b>Descripción</b>
Rebanada	Concentrar la atención en un subconjunto de dimensiones	Reemplazar una dimensión con un valor de miembro individual o con un resumen de sus valores de medida
Dado	Enfocar la atención en un subconjunto de valores del miembro	Sustituir una dimensión con un subconjunto de miembros
Descenso	Obtener más detalles acerca de una dimensión	Navegar de un nivel más general a un nivel más específico de una dimensión jerárquica
Ascenso	Resumir detalles sobre una dimensión	Navegar de un nivel más específico a un nivel más general de una dimensión jerárquica
Pivote	Permitir que se presente un cubo de datos en un orden visualmente atractivo	Reordenar las dimensiones de un cubo de datos

## 16.3 Soporte de DBMS relacional para data warehouse

El modelo multidimensional de los datos descrito en la sección anterior se implementó originalmente con el propósito particular de almacenamiento para cubos de datos. Estos mecanismos de almacenamiento multidimensional soportan la definición, manipulación y optimización de grandes cubos de datos. Como consecuencia del dominio comercial de la tecnología de base de datos relacional, sólo era cuestión de tiempo antes de que los DBMSs relacionales ofrecieran soporte para los datos multidimensionales. En años recientes, los principales fabricantes de DBMS han hecho fuertes inversiones en investigación y desarrollo para soportar los datos multidimensionales. Gracias al nivel de inversión y al poder de mercado de los fabricantes de DBMS relacional, la mayoría de los data warehouse ahora usan DBMSs relacionales, por lo menos en parte.

Esta sección presenta características de los DBMSs relacionales para soportar los datos multidimensionales. Las características incluyen planteamientos de modelado de datos, representación de dimensión, extensiones para la cláusula GROUP BY, vistas materializadas con reescritura de consultas y estructuras de almacenamiento y técnicas de optimización especializadas. Para proporcionar un contexto específico para las características, algunos ejemplos usan Oracle 10g SQL.

### 16.3.1 Modelado de datos relacionales para datos multidimensionales

Al utilizar una base de datos relacional para un data warehouse, se necesita una nueva técnica de modelado de datos a fin de representarlos de manera multidimensional. Un esquema de estrella es una representación del modelado de datos de cubos de datos multidimensionales. En una base de datos relacional, un diagrama de esquema de estrella se ve como una estrella con una tabla central grande en su centro, denominada tabla de hechos, que se vincula con múltiples tablas de dimensión de manera radial. La tabla de hechos almacena datos numéricos (hechos), como los resultados de las ventas, en tanto que las tablas de dimensión almacenan datos descriptivos correspondientes a dimensiones individuales del cubo de datos, como producto, ubicación y tiempo. Hay una relación 1-M de cada tabla de dimensión con la tabla de hechos.

La figura 16.9 muestra un esquema de estrella ERD para el ejemplo de ventas que se presentó en la sección 16.2. Este ERD consiste en cuatro tipos de entidad de dimensión: *Item*, *Customer*, *Store* y *TimeDim*, junto con un tipo de entidad de hecho llamado *Sales*. Al convertirse a un diseño de tabla, la tabla *Sales* tiene llaves foráneas para cada tabla de dimensión (*Item*, *Customer*, *Store* y *TimeDim*). El tipo de entidad *Item* proporciona datos para la dimensión *Product* que se muestra en los ejemplos de la sección 16.2, mientras que el tipo de entidad *Store* proporciona datos para la dimensión *Location*.

El ERD de ventas en tienda de la figura 16.9, ofrece detalles finos para un data warehouse. El ERD de ventas de tienda ofrece detalles del cliente, tienda y artículo individual. Este nivel de detalle no es necesario para soportar los cubos de datos de ventas que se presentan en la sección 16.2. Sin embargo, un nivel de detalle de profundidad ofrece flexibilidad para soportar análisis no anticipados, así como aplicaciones de minería de datos. Este nivel de detalle profundo puede copiar datos en las bases de datos operacionales, aunque la representación del data warehouse puede diferir sustancialmente, dada la orientación al sujeto del data warehouse y la depuración e integración realizadas en los datos de origen.

#### Variaciones al esquema de estrella

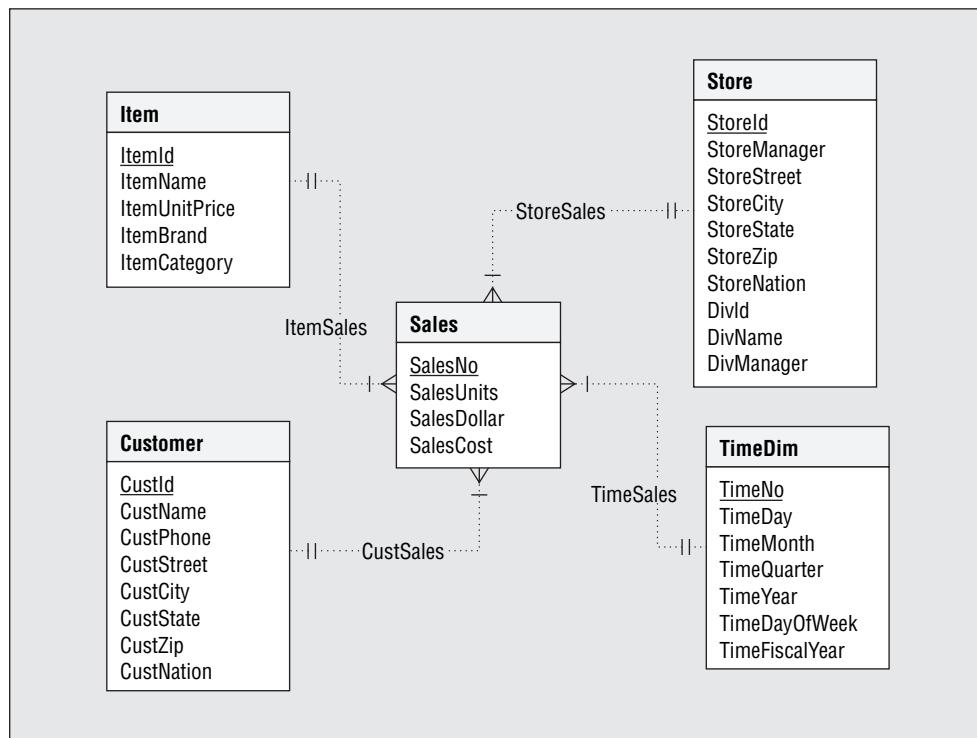
El esquema de estrella de la figura 16.9 representa sólo un proceso empresarial individual para el rastreo de las ventas. Puede requerirse esquemas de estrella adicionales sólo para otros procesos, como envíos y compras. Para procesos empresariales relacionados que comparten algunas tablas de dimensión, un esquema de estrella puede extenderse en un esquema de constelación con tipos de entidad de factor múltiple, como se muestra en la figura 16.10. Al convertirse en un diseño de tabla, el tipo de entidad *Inventory* se convierte en una tabla de hechos y las relaciones 1-M se convierten en llaves foráneas en la tabla de hechos. El tipo de entidad *Inventory* agrega

**esquema de estrella**  
representación de modelado de datos para bases de datos multidimensionales. En una base de datos relacional, un esquema de estrella tiene una tabla de hechos en el centro relacionada con múltiples tablas de dimensión en relaciones 1-M.

**esquema de constelación**  
representación del modelado de datos para bases de datos multidimensionales. En una base de datos relacional, un esquema de constelación contiene múltiples tablas de hechos en el centro relacionadas con tablas de dimensión. Normalmente, las tablas de hechos comparten algunas tablas de dimensión.

**FIGURA 16.9**

**Esquema de estrella**  
ERD para el ejemplo  
de ventas de la tienda



varias medidas, incluida la cantidad disponible de un artículo, el costo del artículo y la cantidad devuelta. Todas las tablas de dimensión se comparten entre ambas tablas de hechos, excepto para las tablas *Supplier* y *Customer*.

Las tablas de hechos por lo general son normalizadas, mientras que las tablas de dimensión a menudo no están en la tercera forma normal. Por ejemplo, el tipo de entidad *Store* en las figuras 16.9 y 16.10 no está en 3NF porque *DivId* determina *DivName* y *DivManager*. Por lo regular, no es necesario normalizar las tablas de dimensión para evitar anomalías de almacenamiento porque generalmente son estables y pequeñas. La naturaleza de un data warehouse indica que deberían diseñarse tablas de dimensión para la recuperación, no para la actualización. El desempeño de la recuperación se mejora al eliminar las operaciones de enlace que se necesitarían para combinar tablas de dimensión totalmente normalizadas.

Cuando las tablas de dimensión son pequeñas, la desnormalización sólo ofrece una pequeña ganancia en el desempeño de la recuperación. Por consiguiente, es común ver tablas en pequeña dimensión normalizadas, como se aprecia en la figura 16.11. Esta variación se conoce como **esquema de copo de nieve** porque múltiples niveles de tablas de dimensión rodean la tabla de hechos. En el caso de las tablas *Customer* e *Item*, la normalización completa quizás no sea una buena idea porque estas tablas pueden contener muchas filas.

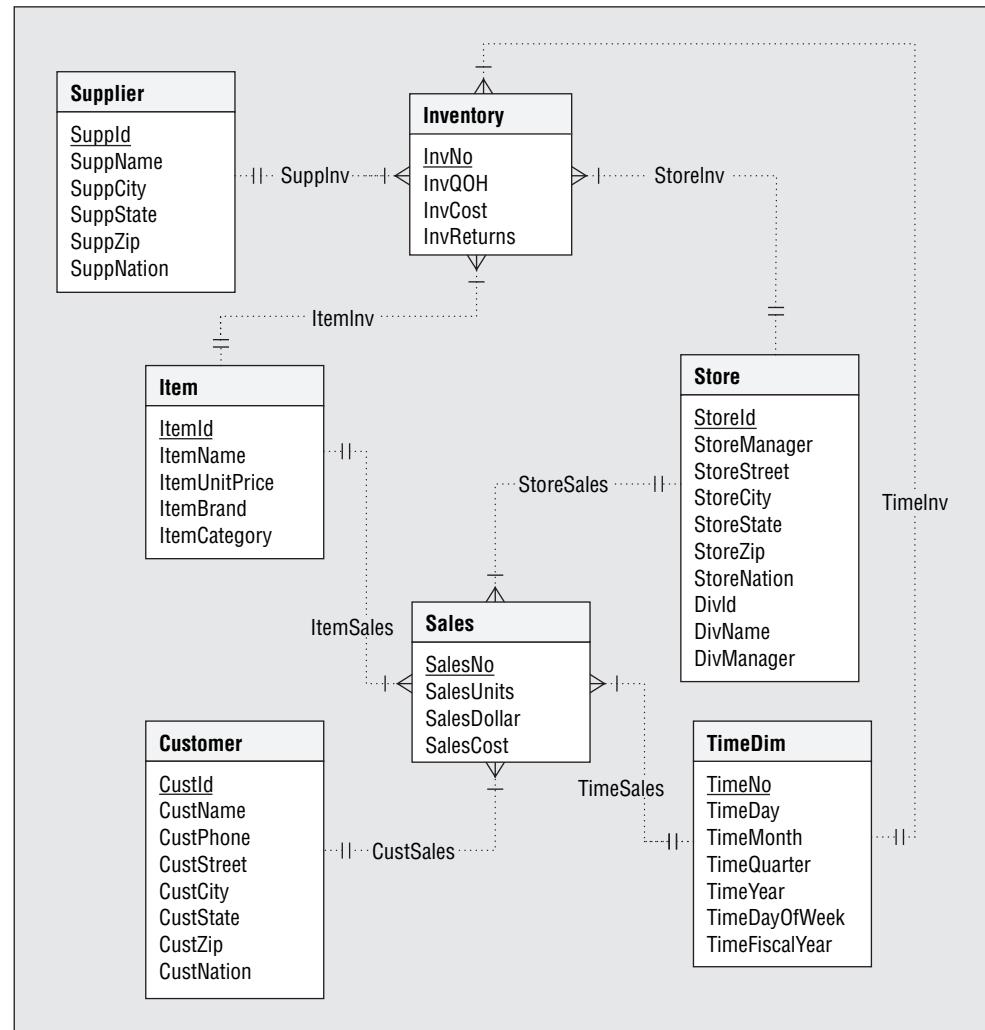
El esquema de estrella y sus variaciones requieren relaciones 1-M entre las tablas de dimensión y la tabla de hechos. El uso de relaciones 1-M simplifica la formulación de consultas y soporta las técnicas de optimización que estudiamos en la sección 16.3.5. En ocasiones, los datos de origen tienen excepciones que implican relaciones M-N, no 1-M. Por ejemplo, si la tabla de hechos *Sales* se deriva de las facturas de clientes, algunas facturas pueden implicar múltiples clientes, como compañeros de cuarto o cónyuges. En seguida se explican dos maneras de revisar el esquema de estrella para relaciones M-N.

- Si hay una cantidad pequeña y fija de clientes posibles, puede hacerse un ajuste simple en la tabla de hechos o de dimensiones. Es posible agregar múltiples columnas a la tabla de hechos o de dimensiones para permitir más de un cliente. Por ejemplo, la tabla *Customer* puede tener una columna adicional *CustId2* para identificar a un segundo cliente opcional en la factura.

### esquema de copo de nieve

representación del modelado de datos para bases de datos multidimensionales. En una base de datos relacional, un esquema de copo de nieve tiene niveles múltiples de tablas de dimensión en relación con una o más tablas de hechos. Debe considerar el uso del esquema de copo de nieve en lugar del esquema de estrella para tablas de poca dimensión que no estén en 3NF.

**FIGURA 16.10**  
Esquema de constelación ERD para el ejemplo de inventario de ventas

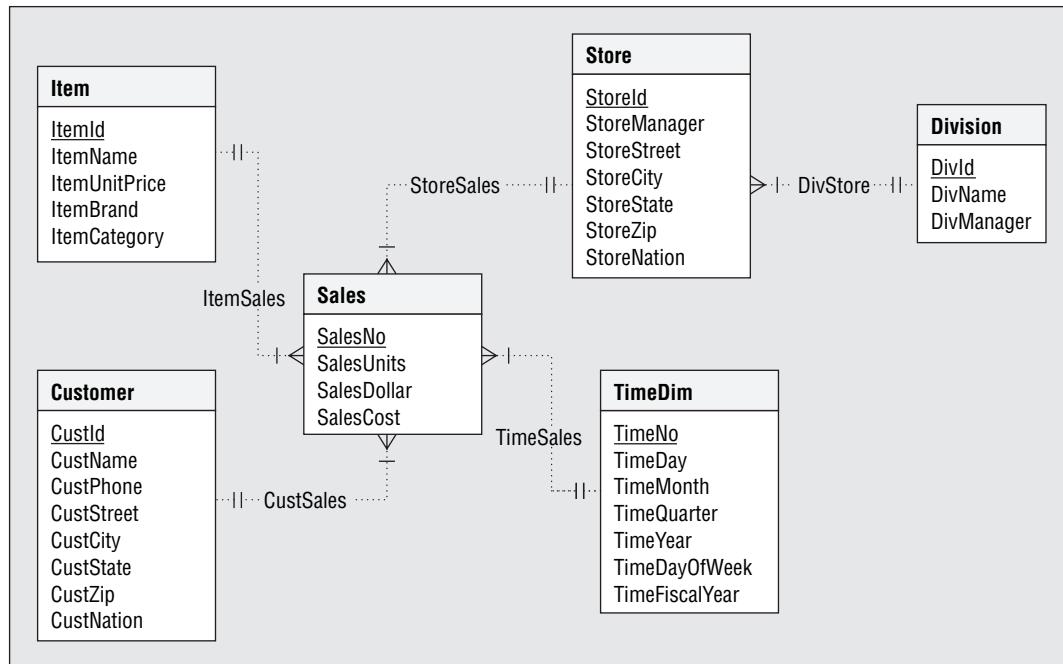


- La representación es más difícil para el caso en que haya grupos de clientes para una factura. Puede añadirse una tabla de grupo de clientes con una tabla asociativa que conecte al grupo de clientes y la tabla *Customer* a través de relaciones 1-M.

#### *Representación temporal en los esquemas de estrella*

La representación temporal es un aspecto crucial para los data warehouse porque la mayoría de las consultas usan el tiempo en las condiciones. El principal uso del tiempo es el registro de la cantidad de hechos. La representación más sencilla es un tipo de datos de estampa de tiempo para una columna en una tabla de hechos. Muchos data warehouse utilizan una llave foránea en una tabla de dimensión de tiempo en lugar de una columna de estampa de tiempo, como se muestra en las figuras 16.9 a 16.11. El uso de una tabla de dimensión de tiempo soporta una conveniente representación de las características de calendario específicas para la organización, como vacaciones, años fiscales y número de semanas que no están representadas en los tipos de datos de estampa de tiempo. La granularidad de la tabla de dimensión de tiempo por lo general se presenta en días. Si también se requiere la hora del día para una tabla de hechos, puede agregarse como una columna en la tabla de hechos a fin de aumentar la llave foránea para la tabla de tiempo.

La mayor parte de las tablas de hechos implican el tiempo representado como una llave foránea para la tabla de tiempo con el aumento para la fecha del día si es necesaria. Por lo que respecta a las tablas de hechos que implican operaciones internacionales, es posible utilizar dos

**FIGURA 16.11** Esquema de copo de nieve ERD para el ejemplo de ventas de tienda

representaciones temporales (llaves foráneas de la tabla de tiempo junto con columnas opcionales para la hora del día) para registrar la hora en los sitios de origen y destino. Una variación identificada por Kimball (2003) es la tabla de hechos acumulativos que registra el estado de múltiples eventos en vez de un solo evento. Por ejemplo, una tabla de hechos que contiene una instantánea del procesamiento de pedidos podría incluir fecha del pedido, fecha de envío, fecha de entrega, fecha de pago y otras. Cada columna de cantidad de eventos puede representarse por medio de una llave foránea para la tabla de tiempo, junto con una columna de hora del día en caso de ser necesario.

Para las tablas de dimensión, la representación temporal implica el nivel de integridad histórica, un aspecto para las actualizaciones de las tablas de dimensión. Cuando se actualiza una fila de dimensión, las filas de la tabla de hechos relacionadas ya no son históricamente precisas. Por ejemplo, si la columna ciudad de una fila de clientes cambia, las filas de ventas relacionadas ya no son históricamente precisas. Con el fin de preservar la integridad histórica, las filas de ventas relacionadas deben apuntar hacia una versión más antigua de la fila de clientes. Kimball (abril de 1996) presenta tres alternativas para la integridad histórica:

- **Tipo I:** Sobrescribe valores antiguos con los datos modificados. Este método no ofrece integridad histórica.
- **Tipo II:** Usa un número de versión para aumentar la llave primaria de una tabla de dimensiones. Para cada cambio en una fila de dimensiones, inserte una fila en la tabla de dimensiones con un número de versión más alto. Por ejemplo, para manejar el cambio en la columna de ciudad, hay una fila nueva en la tabla *Customer* con el mismo número de cliente pero un número de versión mayor que la fila anterior. Además del número de versión, se necesitan columnas adicionales para registrar la fecha de inicio efectiva y la fecha de terminación efectiva para cada columna histórica.
- **Tipo III:** Usa columnas adicionales para mantener un historial fijo. Por ejemplo, con objeto de mantener un historial de la ciudad actual y los dos cambios previos de ciudad, pueden almacenarse tres columnas de ciudad en la tabla *Customer* (*CustCityCurr*, *CustCityPrev*, *CustCityPast*), junto con seis columnas de fecha asociadas (dos columnas de fecha por columna de valor histórico) para registrar las fechas efectivas.

**FIGURA 16.12**  
Alternativas para la integridad dimensional histórica de *CustCity*

Type II Representation	Type III Representation
Customer	Customer
<u>CustId</u>	<u>CustId</u>
<u>VersionNo</u>	<u>CustName</u>
<u>CustName</u>	<u>CustPhone</u>
<u>CustPhone</u>	<u>CustStreet</u>
<u>CustStreet</u>	<u>CustCityCurr</u>
<u>CustCity</u>	<u>CustCityCurrBegEffDate</u>
<u>CustCityBegEffDate</u>	<u>CustCityCurrEndEffDate</u>
<u>CustCityEndEffDate</u>	<u>CustCityPrev</u>
<u>CustState</u>	<u>CustCityPrevBegEffDate</u>
<u>CustZip</u>	<u>CustCityPrevEndEffDate</u>
<u>CustNation</u>	<u>CustCityPast</u>
	<u>CustCityPastBegEffDate</u>
	<u>CustCityPastEndEffDate</u>
	<u>CustState</u>
	<u>CustZip</u>
	<u>CustNation</u>

La figura 16.12 muestra las alternativas tipo II y tipo III para la columna *CustCity*. La alternativa tipo II implica múltiples filas para el mismo cliente, pero se representa el historial entero.

La alternativa tipo III implica sólo una fila individual para cada cliente, pero únicamente puede representarse un historial limitado.

### 16.3.2 Representación de dimensión

El esquema de estrella y sus variaciones no proporcionan una representación explícita de las dimensiones jerárquicas porque las tablas de tiempo no definen las relaciones jerárquicas entre los niveles de una dimensión. Dado que la definición de dimensión es importante para soportar las operaciones del cubo de datos, así como las técnicas de optimización para la reescritura de consultas (véase sección 16.3.4), muchos fabricantes de DBMS relacional han creado extensiones de SQL propietarias para las dimensiones. Esta sección revisa el enunciado de Oracle CREATE DIMENSION para indicar los tipos de extensiones que se pueden encontrar en los DBMSs relacionales.

El enunciado de Oracle CREATE DIMENSION soporta la especificación de niveles, jerarquías y restricciones para una dimensión.<sup>1</sup> La primera parte de una declaración de dimensión implica la especificación de niveles. Para dimensiones planas (no jerárquicas), solamente hay un nivel individual en una dimensión. No obstante, muchas dimensiones implican niveles múltiples,

#### EJEMPLO 16.1

#### Enunciado CREATE DIMENSION de Oracle para la dimensión *StoreDim* con la especificación de niveles

```
CREATE DIMENSION StoreDim
  LEVEL StoreId      IS Store.StoreId
  LEVEL City         IS Store.StoreCity
  LEVEL State        IS Store.StoreState
  LEVEL Zip          IS Store.StoreZip
  LEVEL Nation       IS Store.StoreNation ;
```

<sup>1</sup> No ponga líneas en blanco en los enunciados CREATE DIMENSION. El compilador SQL de Oracle genera mensajes de error cuando encuentra líneas en blanco en estos enunciados.

como se ilustra en el ejemplo 16.1 para la dimensión *StoreDim*. Cada nivel corresponde a una columna de la tabla de origen *Store*.

La siguiente parte de un enunciado CREATE DIMENSION implica la especificación de jerarquías. El enunciado CREATE DIMENSION de Oracle soporta dimensiones con múltiples jerarquías, como se muestra en el ejemplo 16.2. La especificación de una jerarquía se dirige del nivel más detallado al nivel más general. Las palabras clave CHILD OF indican las relaciones de jerarquía directa en una dimensión.

### EJEMPLO 16.2

#### **Enunciado CREATE DIMENSION de Oracle para la dimensión *StoreDim* con la especificación de niveles y jerarquías**

```
CREATE DIMENSION StoreDim
  LEVEL StoreId      IS Store.StoreId
  LEVEL City         IS Store.StoreCity
  LEVEL State        IS Store.StoreState
  LEVEL Zip          IS Store.StoreZip
  LEVEL Nation       IS Store.StoreNation
  HIERARCHY CityRollup  (
    StoreId CHILD OF
    City    CHILD OF
    State   CHILD OF
    Nation  )
  HIERARCHY ZipRollup  (
    StoreId CHILD OF
    Zip     CHILD OF
    State   CHILD OF
    Nation  );
```

El enunciado CREATE DIMENSION de Oracle soporta dimensiones con niveles a partir de múltiples tablas de origen. Esta característica se aplica a las tablas de dimensiones normalizadas en esquemas de copo de nieve. El ejemplo 16.3 aumenta el ejemplo 16.2 con la inclusión de un nivel adicional (*DivId*) junto con una jerarquía adicional que contiene el nuevo nivel. En la especificación de nivel, el nivel *DivId* hace referencia a la tabla *Division*. En la jerarquía *DivisionRollup*, la cláusula JOIN KEY indica un enlace entre las tablas *Store* y *Division*. Cuando una jerarquía tiene niveles de más de una tabla de origen, se requiere la cláusula JOIN KEY al final de la especificación de jerarquías.

### EJEMPLO 16.3

#### **Enunciado CREATE DIMENSION de Oracle para la dimensión *StoreDim* con el uso de múltiples tablas de origen**

```
CREATE DIMENSION StoreDim
  LEVEL StoreId      IS Store.StoreId
  LEVEL City         IS Store.StoreCity
  LEVEL State        IS Store.StoreState
  LEVEL Zip          IS Store.StoreZip
  LEVEL Nation       IS Store.StoreNation
  LEVEL DivId        IS Division.DivId
  HIERARCHY DivisionRollup  (
    StoreId CHILD OF
    Zip     CHILD OF
    State   CHILD OF
    Nation  );
  HIERARCHY CityRollup  (
    StoreId CHILD OF
    City    CHILD OF
    State   CHILD OF
    Nation  );
  HIERARCHY ZipRollup  (
    StoreId CHILD OF
    Zip     CHILD OF
    State   CHILD OF
    Nation  );
```

```

HIERARCHY CityRollup  (
    StoreId CHILD OF
    City     CHILD OF
    State   CHILD OF
    Nation  )
HIERARCHY ZipRollup  (
    StoreId CHILD OF
    Zip      CHILD OF
    State   CHILD OF
    Nation  )
HIERARCHY DivisionRollup  (
    StoreId CHILD OF
    DivId
    JOIN KEY Store.DivId REFERENCES DivId );

```

La parte final de un enunciado CREATE DIMENSION implica la especificación de restricciones. La cláusula ATTRIBUTE define relaciones de dependencia funcional que implican niveles de dimensión y columnas de no origen en tablas de dimensiones. El ejemplo 16.4 muestra cláusulas ATTRIBUTE para las columnas de no origen en la tabla *Division*.

**EJEMPLO 16.4****Enunciado CREATE DIMENSION de Oracle para la dimensión *StoreDim* con el uso de cláusulas ATTRIBUTE para las restricciones**

```

CREATE DIMENSION StoreDim
    LEVEL StoreId      IS Store.StoreId
    LEVEL City         IS Store.StoreCity
    LEVEL State        IS Store.StoreState
    LEVEL Zip          IS Store.StoreZip
    LEVEL Nation       IS Store.StoreNation
    LEVEL DivId        IS Division.DivId
    HIERARCHY CityRollup  (
        StoreId CHILD OF
        City     CHILD OF
        State   CHILD OF
        Nation  )
    HIERARCHY ZipRollup  (
        StoreId CHILD OF
        Zip      CHILD OF
        State   CHILD OF
        Nation  )
    HIERARCHY DivisionRollup  (
        StoreId CHILD OF
        DivId
        JOIN KEY Store.DivId REFERENCES DivId )
    ATTRIBUTE DivId DETERMINES Division.DivName
    ATTRIBUTE DivId DETERMINES Division.DivManager ;

```

En el ejemplo 16.4, las cláusulas DETERMINES son redundantes con la restricción de llave primaria para la tabla *Division*. Las cláusulas DETERMINES se muestran en el ejemplo 16.4 para reforzar las restricciones soportadas por las declaraciones de llaves primarias. Las

cláusulas DETERMINES se requieren para restricciones que no corresponden con las restricciones de llaves primarias para permitir la optimización de las consultas. Por ejemplo, si cada código postal se asocia con un estado, debería utilizarse una cláusula DETERMINES para permitir optimizaciones que implican columnas de código postal y estado.

### 16.3.3 Extensiones para la cláusula GROUP BY para datos multidimensionales

Hay disponibles nuevas capacidades de resumen en la cláusula GROUP BY, comenzando en SQL: 1999 y continuando con SQL: 2003. Estas características son un intento por unificar la proliferación de extensiones propietarias de cubo de datos, aunque no ponen en evidencia la necesidad de herramientas visuales para soportar directamente las operaciones del cubo de datos. Las extensiones implican la capacidad de producir totales sumatorios (operadores CUBE y ROLLUP), así como la especificación más precisa de las columnas de agrupación (operadores GROUPING SETS). Esta sección describe estas nuevas partes de la cláusula GROUP BY usando Oracle 10g como ejemplo de DBMS que implementa las características estándar de SQL.

#### *Operador CUBE*

**operador CUBE**  
operador que aumenta el resultado normal GROUP BY con todas las combinaciones de subtotales. El operador CUBE es apropiado para resumir columnas de dimensiones independientes más que columnas que representen diferentes niveles de una sola dimensión.

La cláusula del operador CUBE produce todas las combinaciones de subtotales además de los totales normales que se muestran en una cláusula GROUP BY. Debido a que se generan todos los subtotales posibles, el operador CUBE es apropiado para resumir columnas de dimensiones independientes más que columnas que representen diferentes niveles de la misma dimensión. Por ejemplo, el operador CUBE podría ser apropiado para generar subtotales para todas las combinaciones de mes, estado de la tienda y marca del artículo. En contraste, una operación CUBE tendría interés limitado para mostrar todos los subtotales posibles de año, mes y día, como resultado de la jerarquía en la dimensión de tiempo.

Para ilustrar el operador CUBE, el ejemplo 16.5 presenta el enunciado SELECT con una cláusula GROUP BY que contiene sólo dos columnas. En el resultado aparecen sólo seis filas, de modo que puede comprenderse con facilidad el efecto del operador CUBE. Con dos valores en la columna *StoreZip* y tres valores en la columna *TimeMonth*, el número de combinaciones

#### EJEMPLO 16.5

#### Cláusula GROUP BY y resultado parcial sin subtotales

```
SELECT StoreZip, TimeMonth, SUM(SalesDollar) AS SumSales
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId AND
       Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear = 2005
 GROUP BY StoreZip, TimeMonth;
```

StoreZip	TimeMonth	SumSales
80111	1	10000
80111	2	12000
80111	3	11000
80112	1	9000
80112	2	11000
80112	3	15000

**EJEMPLO 16.6  
(Oracle)****Cláusula GROUP BY y resultado con subtotales producidos por el operador CUBE**

```
SELECT StoreZip, TimeMonth, SUM(SalesDollar) AS SumSales
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear = 2005
 GROUP BY CUBE(StoreZip, TimeMonth);
```

StoreZip	TimeMonth	SumSales
80111	1	10000
80111	2	12000
80111	3	11000
80112	1	9000
80112	2	11000
80112	3	15000
80111		33000
80112		35000
	1	19000
	2	23000
	3	26000
		68000

del subtotal es seis (dos subtotales *StoreZip*, tres subtotales *TimeMonth* y un gran total), como se aprecia en el ejemplo 16.6. Los valores en blanco en el resultado representan un resumen sobre todos los valores posibles de la columna. Por ejemplo, la fila <80111, -, 33000> representa las ventas totales en el código postal 80111 en todos los meses (– representa un valor sin interés para el mes).

Con más de dos columnas de agrupación, se hace más difícil entender el operador CUBE. Los ejemplos 16.7 y 16.8 extienden los ejemplos 16.5 y 16.6 con una columna de agrupación adicional (*TimeYear*). El número de filas en el resultado aumenta de 12 filas en el resultado del ejemplo 16.7 sin el operador CUBE a 36 filas en el resultado del ejemplo 16.8 con el operador CUBE. Para tres columnas de agrupación con valores únicos *M*, *N* y *P*, el número máximo de filas de subtotal producidas por el operador CUBE es  $M + N + P + M * N + M * P + N * P + 1$ . Como el número de filas de subtotal crece sustancialmente con el número de columnas agrupadas y los valores únicos por columna, debería utilizarse muy poco el operador CUBE cuando hay más de tres columnas agrupadas.

**EJEMPLO 16.7****Cláusula GROUP BY con tres columnas de agrupación y el resultado parcial sin subtotales**

```
SELECT StoreZip, TimeMonth, TimeYear, SUM(SalesDollar) AS SumSales
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear BETWEEN 2005 AND 2006
 GROUP BY StoreZip, TimeMonth, TimeYear;
```

StoreZip	TimeMonth	TimeYear	SumSales
80111	1	2005	10000
80111	2	2005	12000
80111	3	2005	11000
80112	1	2005	9000
80112	2	2005	11000
80112	3	2005	15000
80111	1	2006	11000
80111	2	2006	13000
80111	3	2006	12000
80112	1	2006	10000
80112	2	2006	12000
80112	3	2006	16000

**EJEMPLO 16.8  
(Oracle)****Cláusula GROUP BY con tres columnas de agrupación y el resultado con subtotales producidos por el operador CUBE**

```

SELECT StoreZip, TimeMonth, TimeYear, SUM(SalesDollar) AS SumSales
FROM Sales, Store, TimeDim
WHERE Sales.StoreId = Store.StoreId
      AND Sales.TimeNo = TimeDim.TimeNo
      AND (StoreNation = 'USA' OR StoreNation = 'Canada')
      AND TimeYear BETWEEN 2005 AND 2006
GROUP BY CUBE(StoreZip, TimeMonth, TimeYear);

```

StoreZip	TimeMonth	TimeYear	SumSales
80111	1	2005	10000
80111	2	2005	12000
80111	3	2005	11000
80112	1	2005	9000
80112	2	2005	11000
80112	3	2005	15000
80111	1	2006	11000
80111	2	2006	13000
80111	3	2006	12000
80112	1	2006	10000
80112	2	2006	12000
80112	3	2006	16000
80111	1		21000
80111	2		25000
80111	3		23000
80112	1		19000
80112	2		22000
80112	3		31000
80111		2005	33000
80111		2006	36000
80112		2005	35000

StoreZip	TimeMonth	TimeYear	SumSales
80112		2006	38000
	1	2005	19000
	2	2005	23000
	3	2005	26000
	1	2006	21000
	2	2006	25000
	3	2006	28000
80111			69000
80112			73000
	1		40000
	2		48000
	3		54000
		2005	68000
		2006	74000
			142000

El operador CUBE no es un operador primitivo. El resultado de una operación CUBE puede producirse usando varios enunciados SELECT conectados por medio del operador UNION, como se muestra en el ejemplo 16.9. Los enunciados SELECT adicionales generan subtotales para cada combinación de columnas agrupadas. Con dos columnas agrupadas, se necesitan tres enunciados SELECT adicionales para generar los subtotales. Con  $N$  columnas agrupadas, son necesarios  $2^N - 1$  enunciados SELECT adicionales. Obviamente, es mucho más fácil escribir el operador CUBE que un gran número de enunciados SELECT.

### EJEMPLO 16.9

#### Reescritura del ejemplo 16.6 sin usar el operador CUBE

En cada enunciado SELECT adicional, un valor predeterminado (0 para las columnas numéricas y '' para columnas de texto) reemplaza la columna en donde no se generan totales.

```

SELECT StoreZip, TimeMonth, SUM(SalesDollar) AS SumSales
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear = 2005
 GROUP BY StoreZip, TimeMonth
 UNION
SELECT StoreZip, 0, SUM(SalesDollar) AS SumSales
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear = 2005
 GROUP BY StoreZip
 UNION
SELECT '', TimeMonth, SUM(SalesDollar) AS SumSales
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId

```

```

        AND Sales.TimeNo = TimeDim.TimeNo
        AND (StoreNation = 'USA' OR StoreNation = 'Canada')
        AND TimeYear = 2005
    GROUP BY TimeMonth
UNION
SELECT '', 0, SUM(SalesDollar) AS SumSales
    FROM Sales, Store, TimeDim
   WHERE Sales.StoreId = Store.StoreId
        AND Sales.TimeNo = TimeDim.TimeNo
        AND (StoreNation = 'USA' OR StoreNation = 'Canada')
        AND TimeYear = 2005;

```

**operador ROLLUP**

operador que aumenta el resultado normal de GROUP BY con un conjunto parcial de subtotales. El operador ROLLUP es apropiado para resumir niveles de una jerarquía de dimensión.

*Operador ROLLUP*

El operador SQL ROLLUP proporciona una capacidad similar al operador de ascenso para cubos de datos. El operador de ascenso para cubos de datos produce totales para más partes generales de una jerarquía de dimensión. El operador SQL ROLLUP produce subtotales para cada subconjunto ordenado de columnas agrupadas para simular los efectos del operador de ascenso para cubos de datos. Por ejemplo, la operación de *SQL ROLLUP (TimeYear, TimeQuarter, TimeMonth, TimeDay)* produce subtotales para los subconjuntos de columna <TimeYear, TimeQuarter, TimeMonth>, <TimeYear, TimeQuarter>, <TimeYear>, así como el gran total. Como implica este ejemplo, el orden de las columnas en una operación ROLLUP es significativo.

Como indica el párrafo anterior, el operador ROLLUP produce sólo un conjunto parcial de subtotales para las columnas en una cláusula GROUP BY. Los ejemplos 16.10 y 16.11 demuestran el operador ROLLUP con el fin de contrastarlo con el operador CUBE en el ejemplo 16.6.

**EJEMPLO 16.10**  
**(Oracle)****Cláusula GROUP BY y resultado con subtotales producidos por el operador ROLLUP**

This example should be compared with Example 16.6 to understand the difference between the CUBE and ROLLUP operators.

```

SELECT StoreZip, TimeMonth, SUM(SalesDollar) AS SumSales
    FROM Sales, Store, TimeDim
   WHERE Sales.StoreId = Store.StoreId
        AND Sales.TimeNo = TimeDim.TimeNo
        AND (StoreNation = 'USA' OR StoreNation = 'Canada')
        AND TimeYear = 2005
    GROUP BY ROLLUP(StoreZip, TimeMonth);

```

StoreZip	TimeMonth	SumSales
80111	1	10000
80111	2	12000
80111	3	11000
80112	1	9000
80112	2	11000
80112	3	15000
80111		33000
80112		35000
		68000

Nótese que el ejemplo 16.10 contiene tres filas subtotales comparadas con seis filas subtotales. Este ejemplo debe compararse con el ejemplo 16.6 para entender la diferencia entre los operadores CUBE y ROLLUP en el ejemplo 16.6 con el operador CUBE. En el ejemplo 16.11, se producen subtotales para los valores en las combinaciones de columna *<StoreZip, TimeMonth>*, *<StoreZip>*, y el gran total. En el ejemplo 16.8, con el operador CUBE, también se producen subtotales para los valores en las combinaciones de columna *<StoreZip, TimeYear>*, *<TimeMonth, TimeYear>*, *<TimeMonth>* y *<TimeYear>*. Por lo tanto, el operador ROLLUP produce muchas menos filas de subtotal para el operador CUBE conforme se incrementa el número de columnas agrupadas y los valores únicos por columna.

**EJEMPLO 16.11**  
**(Oracle)**
**Cláusula GROUP BY con tres columnas de agrupación y el resultado con subtotales producidos por el operador ROLLUP**

Este ejemplo debe compararse con el ejemplo 16.8 para entender la diferencia entre los operadores CUBE y ROLLUP.

```
SELECT StoreZip, TimeMonth, TimeYear, SUM(SalesDollar) AS SumSales
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear BETWEEN 2005 AND 2006
 GROUP BY ROLLUP(StoreZip, TimeMonth, TimeYear);
```

StoreZip	TimeMonth	TimeYear	SumSales
80111	1	2005	10000
80111	2	2005	12000
80111	3	2005	11000
80112	1	2005	9000
80112	2	2005	11000
80112	3	2005	15000
80111	1	2006	11000
80111	2	2006	13000
80111	3	2006	12000
80112	1	2006	10000
80112	2	2006	12000
80112	3	2006	16000
80111	1		21000
80111	2		25000
80111	3		23000
80112	1		19000
80112	2		22000
80112	3		31000
80111			69000
80112			73000
			142000

Al igual que el operador CUBE, el operador ROLLUP no es un operador primitivo. El resultado de una operación ROLLUP puede producirse utilizando varios enunciados SELECT conectados por el operador UNION, como se muestra en el ejemplo 16.12. Los enunciados SELECT adicionales generan subtotales para cada subconjunto ordenado de columnas agrupadas.

Con tres columnas agrupadas se necesitan tres enunciados SELECT adicionales para generar los subtotales. Con  $N$  columnas agrupadas, se necesitan  $N$  enunciados SELECT adicionales. Es evidente que es mucho más fácil escribir el operador ROLLUP que un gran número de enunciados SELECT adicionales.

### EJEMPLO 16.12

#### Reescritura del ejemplo 16.11 sin usar el operador ROLLUP

En cada enunciado SELECT adicional, un valor predeterminado (0 para las columnas numéricas y "" para columnas de texto) reemplaza la columna en donde no se generan totales.

```

SELECT StoreZip, TimeMonth, TimeYear, SUM(SalesDollar) AS SumSales
    FROM Sales, Store, TimeDim
   WHERE Sales.StoreId = Store.StoreId
         AND Sales.TimeNo = TimeDim.TimeNo
         AND (StoreNation = 'USA' OR StoreNation = 'Canada')
         AND TimeYear BETWEEN 2005 AND 2006
 GROUP BY StoreZip, TimeMonth, TimeYear
 UNION
SELECT StoreZip, TimeMonth, 0, SUM(SalesDollar) AS SumSales
    FROM Sales, Store, TimeDim
   WHERE Sales.StoreId = Store.StoreId
         AND Sales.TimeNo = TimeDim.TimeNo
         AND (StoreNation = 'USA' OR StoreNation = 'Canada')
         AND TimeYear BETWEEN 2005 AND 2006
 GROUP BY StoreZip, TimeMonth
 UNION
SELECT StoreZip, 0, 0, SUM(SalesDollar) AS SumSales
    FROM Sales, Store, TimeDim
   WHERE Sales.StoreId = Store.StoreId
         AND Sales.TimeNo = TimeDim.TimeNo
         AND (StoreNation = 'USA' OR StoreNation = 'Canada')
         AND TimeYear BETWEEN 2005 AND 2006
 GROUP BY StoreZip
 UNION
SELECT "", 0, 0, SUM(SalesDollar) AS SumSales
    FROM Sales, Store, TimeDim
   WHERE Sales.StoreId = Store.StoreId
         AND Sales.TimeNo = TimeDim.No
         AND (StoreNation = 'USA' OR StoreNation = 'Canada')
         AND TimeYear BETWEEN 2005 AND 2006;

```

### operador

#### GROUPING SETS

operador en la cláusula GROUP BY que requiere especificación explícita de las combinaciones de columna. El operador GROUPING SETS es apropiado cuando se requiere control preciso de la agrupación y los subtotales.

### Operador GROUPING SETS

Para una mayor flexibilidad de la que proporcionan los operadores CUBE y ROLLUP, puede emplear el operador GROUPING SETS. Cuando usa el operador GROUPING SETS, especifica explícitamente las combinaciones de columnas para las cuales necesita totales. En contraste, la especificación de subtotales es implícita en los operadores CUBE y ROLLUP. Si no se requiere control explícito, los operadores CUBE y ROLLUP dan una especificación más sucinta.

Para ilustrar el operador GROUPING SETS, los ejemplos anteriores han sido modificados usando el operador GROUPING SETS. En el ejemplo 16.13, el operador GROUPING SETS implica subtotales para las columnas *StoreZip* y *TimeMonth* junto con el gran total expresado por el paréntesis vacío. El subconjunto (*StoreZip*, *TimeMonth*) también debe especificarse porque todas

las combinaciones de columna tienen que especificarse en forma explícita, incluso la agrupación normal sin el operador GROUPING SETS. El ejemplo 16.14 contiene ocho combinaciones de columna para dar el mismo resultado que el ejemplo 16.8 con el CUBE de tres columnas. El ejemplo 16.15 contiene tres combinaciones de columna para dar el mismo resultado que el ejemplo 16.11 con el ROLLUP de tres columnas.

**EJEMPLO 16.13**  
**(Oracle)**
**Cláusula GROUP BY con el uso del operador GROUPING SETS**

Este ejemplo produce el mismo resultado que el ejemplo 16.6.

```
SELECT StoreZip, TimeMonth, SUM(SalesDollar) AS SumSales
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
       AND Sales.TimeNo = TimeDim.TimeNo
       AND (StoreNation = 'USA' OR StoreNation = 'Canada')
       AND TimeYear = 2005
 GROUP BY GROUPING SETS((StoreZip, TimeMonth), StoreZip,
                        TimeMonth, ());
```

**EJEMPLO 16.14**  
**(Oracle)**
**Cláusula GROUP BY con el uso del operador GROUPING SETS**

Este ejemplo produce el mismo resultado que el ejemplo 16.8.

```
SELECT StoreZip, TimeMonth, TimeYear, SUM(SalesDollar) AS SumSales
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
       AND Sales.TimeNo = TimeDim.TimeNo
       AND (StoreNation = 'USA' OR StoreNation = 'Canada')
       AND TimeYear BETWEEN 2005 AND 2006
 GROUP BY GROUPING SETS((StoreZip, TimeMonth, TimeYear),
                        (StoreZip, TimeMonth), (StoreZip, TimeYear),
                        (TimeMonth, TimeYear), StoreZip, TimeMonth, TimeYear, () );
```

**EJEMPLO 16.15**  
**(Oracle)**
**Cláusula GROUP BY con el uso del operador GROUPING SETS**

Este ejemplo produce el mismo resultado que el ejemplo 16.11.

```
SELECT StoreZip, TimeMonth, TimeYear, SUM(SalesDollar) AS SumSales
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
       AND Sales.TimeNo = TimeDim.TimeNo
       AND (StoreNation = 'USA' OR StoreNation = 'Canada')
       AND TimeYear BETWEEN 2005 AND 2006
 GROUP BY GROUPING SETS( (StoreZip, TimeMonth, TimeYear),
                        (StoreZip, TimeMonth), StoreZip, () );
```

El ejemplo 16.16 ilustra una situación en donde es preferible el operador GROUPING SETS que el operador CUBE. Como las columnas *TimeYear* y *TimeMonth* son de la misma jerarquía de dimensión, por lo general no se garantiza un cubo completo. En su lugar, es posible utilizar el operador GROUPING SETS para especificar las combinaciones de columna de las que se necesitan subtotales. Los subtotales que implican *TimeMonth* y *TimeYear* se excluyen en el ejemplo 16.16, pero se incluye en una operación completa de CUBE.

**EJEMPLO 16.16**  
**(Oracle)**
**Cláusula GROUP BY con el uso del operador GROUPING SETS para indicar las combinaciones de columna de las que se necesitan subtotales**

```
SELECT StoreZip, TimeMonth, TimeYear, SUM(SalesDollar) AS SumSales
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
       AND Sales.TimeNo = TimeDim.TimeNo
       AND (StoreNation = 'USA' OR StoreNation = 'Canada')
       AND TimeYear BETWEEN 2005 AND 2006
 GROUP BY GROUPING SETS( (StoreZip, TimeYear, TimeMonth),
                         (StoreZip, TimeYear), (TimeMonth, TimeYear),
                         StoreZip, TimeYear, () );
```

### Variaciones de los operadores CUBE, ROLLUP y GROUPING SETS

Los operadores CUBE, ROLLUP y GROUPING SETS pueden combinarse para dar la mezcla adecuada de especificaciones de agrupación con el operador GROUPING SETS junto con subtotales proporcionados por los operadores ROLLUP y CUBE. Esta lista proporciona algunas de las variaciones posibles cuando se utilizan estos operadores.

- Puede hacerse un CUBE parcial para producir subtotales para un subconjunto de dimensiones independientes. Por ejemplo, la cláusula *GROUP BY TimeMonth, CUBE(ItemBrand, StoreState)* produce totales en los subconjuntos de columna <*TimeMonth, ItemBrand, StoreState*>, <*TimeMonth, ItemBrand*>, <*TimeMonth, StoreState*> y <*TimeMonth*>.
- Puede hacerse un ROLLUP parcial para producir subtotales de columnas de la misma jerarquía de dimensión. Por ejemplo, la cláusula *GROUP BY ItemBrand, ROLLUP(TimeYear, TimeMonth, TimeDay)* produce totales en los subconjuntos de columna <*ItemBrand, TimeYear, TimeMonth, TimeDay*>, <*ItemBrand, TimeYear, TimeMonth*>, <*ItemBrand, TimeYear*> y <*ItemBrand*>.
- Es posible usar columnas compuestas con los operadores CUBE y ROLLUP para saltarse algunos subtotales. Por ejemplo, la cláusula *GROUP BY ROLLUP (TimeYear, (TimeQuarter, TimeMonth), TimeDay)* produce totales en los subconjuntos de columna <*TimeYear, TimeQuarter, TimeMonth, TimeDay*>, <*TimeYear, TimeQuarter, TimeMonth*>, <*TimeYear*> y <>. La columna compuesta <*TimeQuarter, TimeMonth*> se trata como una columna sencilla en la operación ROLLUP.
- Las operaciones CUBE y ROLLUP pueden incluirse en una operación GROUPING SETS. Por ejemplo, la cláusula *GROUP BY GROUPING SETS (ItemBrand, ROLLUP (TimeYear, TimeMonth), StoreState)* produce totales en los subconjuntos de columna <*ItemBrand*>, <*StoreState*>, <*TimeYear, TimeMonth*>, <*TimeYear*> y <>. La operación empaquetada ROLLUP crea subtotales en los subconjuntos de columna <*TimeYear, TimeMonth*>, <*TimeYear*> y <>.

### Otras extensiones para el apoyo a las decisiones

Además de las extensiones GROUP BY, pueden usarse varias funciones agregadas en el enunciado SELECT. A continuación se listan algunas extensiones comunes.

- La función de clasificación soporta consultas para el porcentaje superior e inferior de los resultados.
- Las funciones de razón simplifican la formulación de consultas que comparan valores individuales con los totales de grupo.
- Las funciones para mover totales y promedios permiten facilitar el análisis de datos de la serie de tiempo. La extensión OLAP en SQL:2003 proporciona la cláusula WINDOW para especificar los promedios de movimiento.

### 16.3.4 Vistas materializadas y reescritura de consultas

**vista materializada**

vista almacenada que debe sincronizarse periódicamente con sus datos de origen. Las vistas materializadas soportan el almacenamiento de datos resumidos para respuesta rápida de consultas.

Para soportar las consultas que implican grandes tablas de hechos, los DBMSs relacionales proporcionan vistas materializadas. Una vista materializada es una vista almacenada que debe sincronizarse de manera periódica con sus datos de origen. Las vistas materializadas son atractivas en los data warehouse porque los datos de origen son estables para refrescamientos periódicos que generalmente se realizan durante horas no pico. En contraste, las vistas tradicionales (no materializadas) dominan el procesamiento de la base de datos operacional porque el costo de refrescamiento puede ser alto. Junto con las vistas materializadas, la mayoría de los DBMSs relacionales soportan la sustitución automática de las vistas materializadas para tablas de origen en un proceso conocido como reescritura de consultas. Esta sección ilustra las vistas materializadas utilizando la sintaxis de Oracle 10g y ofrece ejemplos del proceso de reescritura de consultas.

#### *Vistas materializadas en Oracle 10g*

La especificación de una vista materializada en Oracle 10g implica elementos de especificación de la tabla base y de la diagramación de vistas tradicionales, junto con la especificación de las propiedades de la materialización. Debido a que se almacenan las vistas materializadas, la mayor parte de las propiedades de almacenamiento para las tablas base también puede especificarse para las vistas materializadas. Puesto que aquí no nos concentraremos en las propiedades de almacenamiento, no se ilustrarán. La especificación del mapeo es la misma para las vistas tradicionales que para las vistas materializadas. Un enunciado SELECT ofrece el mapeo necesario para poblar una vista materializada. Las propiedades de materialización incluyen

- **Método de refrescamiento (incremental o completo):** Oracle tiene varias restricciones sobre los tipos de vistas materializadas que pueden refrescarse incrementalmente, de modo que aquí no se estudiará el refrescamiento incremental.
- **Tiempo de refrescamiento (sobre pedido o por asignación):** Para la opción sobre pedido, Oracle proporciona el paquete DBMS\_MView con varios procedimientos de refrescamiento para especificar detalles del tiempo de refrescamiento (Refresh, Refresh\_All\_MViews, Refresh\_Dependent).
- **Tiempo de construcción (inmediato o diferido):** Para la opción diferida, los procedimientos de refrescamiento en la DBMS\_MView pueden utilizarse para especificar los detalles de la población de la vista materializada.

Los ejemplos 16.17 a 16.19 ilustran la sintaxis del enunciado CREATE MATERIALIZED VIEW. Estos enunciados parecen similares a los enunciados CREATE VIEW, excepto por las cláusulas de materialización. El tiempo de construcción es inmediato en los ejemplos 16.17 y 16.19, en tanto que es diferido en el ejemplo 16.18. El método de refrescamiento es completo y el tiempo de refrescamiento es sobre pedido en las tres vistas materializadas. El enunciado SELECT después de la palabra clave AS proporciona la diagramación para poblar una vista materializada.

La conciencia del usuario es otra diferencia entre las vistas tradicionales y las vistas materializadas. Para consultas con el uso de bases de datos operacionales, se emplean vistas tradicionales en lugar de tablas base para simplificar la formulación de consultas. Un usuario percibe la base de datos como una vista que escuda al usuario de las complejidades de las tablas base. En contraste, las consultas al data warehouse solicitadas por los usuarios, se usan tablas de hechos y dimensiones. Las tablas de hechos y dimensiones pueden ocultarse por medio de una

**EJEMPLO 16.17  
(Oracle)** **Vista materializada que contiene las ventas para todos los países para los años posteriores a 2003 agrupadas por estado y año**

```

CREATE MATERIALIZED VIEW MV1
BUILD IMMEDIATE
REFRESH COMPLETE ON DEMAND
ENABLE QUERY REWRITE AS
SELECT StoreState, TimeYear, SUM(SalesDollar) AS SUMDollar1
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND TimeYear > 2003
 GROUP BY StoreState, TimeYear;

```

**EJEMPLO 16.18  
(Oracle)** **Vista materializada que contiene las ventas en Estados Unidos en todos los años agrupadas por estado, año y mes**

```

CREATE MATERIALIZED VIEW MV2
BUILD DEFERRED
REFRESH COMPLETE ON DEMAND
ENABLE QUERY REWRITE AS
SELECT StoreState, TimeYear, TimeMonth,
       SUM(SalesDollar) AS SUMDollar2
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND StoreNation = 'USA'
 GROUP BY StoreState, TimeYear, TimeMonth;

```

**EJEMPLO 16.19  
(Oracle)** **Vista materializada que contiene las ventas en Canadá antes de 2004 agrupadas por ciudad, año y mes**

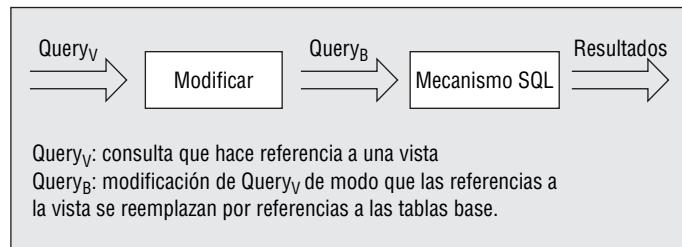
```

CREATE MATERIALIZED VIEW MV3
BUILD IMMEDIATE
REFRESH COMPLETE ON DEMAND
ENABLE QUERY REWRITE AS
SELECT StoreCity, TimeYear, TimeMonth,
       SUM(SalesDollar) AS SUMDollar3
  FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND StoreNation = 'Canada'
   AND TimeYear <= 2003
 GROUP BY StoreCity, TimeYear, TimeMonth;

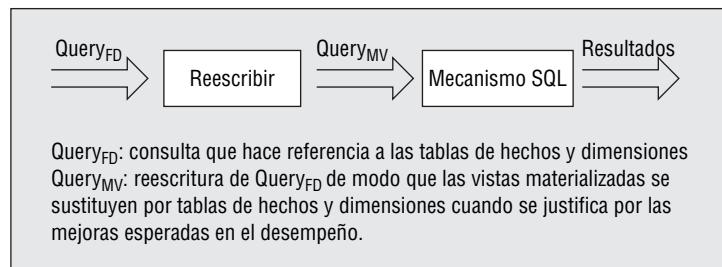
```

herramienta de consulta para simplificar la formulación de consultas. Los usuarios del data warehouse no están conscientes de las vistas materializadas, que sólo son elementos de ayuda para el desempeño administrado por el DBMS. El DBMS proporciona herramientas para ayudar

**FIGURA 16.13**  
Flujo del proceso de modificación de vistas



**FIGURA 16.14**  
Flujo del Proceso de reescritura de consultas



a determinar las vistas materializadas apropiadas y usarlas con el fin de mejorar el desempeño de las consultas en un proceso que recibe el nombre de reescritura de consultas.

#### *Principios de reescritura de consultas*

**reescritura de consultas**  
proceso de sustitución en que una vista materializada reemplaza las referencias a las tablas de hechos y dimensiones en una consulta. El optimizador de consultas evalúa si la sustitución mejorará el desempeño de la consulta original sin la sustitución de la vista materializada.

El proceso de reescritura de consultas para las vistas materializadas revierte el proceso de modificación de consultas para las vistas tradicionales que se presentan en el capítulo 10. Recuerde que el proceso de modificación de consultas (véase la figura 16.13) sustituye las tablas base por vistas, de manera que no se necesita la materialización de las vistas. En contraste, el proceso de reescritura de consultas (véase la figura 16.14) sustituye las vistas materializadas con tablas de hechos y dimensiones con el fin de evitar entrar a grandes tablas de hechos y dimensiones. Sólo se lleva a cabo el proceso de sustitución si se esperan mejoras en el desempeño.

Por lo general, la reescritura de consultas es más compleja que la modificación de consultas porque las reescritura de consultas implica un proceso de sustitución más complicado y requiere del optimizador para evaluar los costos. En ambos procesos el DBMS, y no el usuario, realiza el proceso de sustitución. En el proceso de reescritura de consultas, el optimizador debe evaluar si la sustitución mejorará el desempeño de la consulta original. En el proceso de modificación de consultas, el optimizador de consultas no compara el costo de la consulta modificada con la consulta original porque la modificación generalmente ofrece ahorros sustanciales.

#### *Detalles de la reescritura de consultas*

La reescritura de consultas implica un proceso de coincidencia entre una consulta mediante el uso de tablas de hechos y dimensiones, y una serie de vistas materializadas que contienen datos resumidos. En síntesis, una vista materializada puede proporcionar datos para una consulta si la vista materializada concuerda con las condiciones de fila, columnas de agrupación y funciones agregadas, como se explica a continuación y se resume en la tabla 16.9.

- **Coincidencia de condición de fila:** La reescritura no es posible si una vista materializada contiene más condiciones restrictivas que las condiciones de la vista materializada. Por ejemplo, si una vista materializada tiene las condiciones StoreNation = 'USA' y TimeYear = 2005, pero una consulta sólo tiene la condición StoreNation = 'USA', la vista materializada no puede proporcionar los datos para la consulta.

**TABLA 16.9**

**Resumen de requerimientos de coincidencia para la reescritura de consultas**

Tipo de coincidencia	Requerimientos
Condiciones de fila	Las condiciones de la consulta deben ser por lo menos tan restrictivas como las condiciones de las vistas materializadas.
Detalle de agrupación	Las columnas de agrupación de consultas no deben ser más detalladas que las columnas de agrupación de vista materializada.
Dependencias de agrupación	Las columnas de consulta deben coincidir o poderse derivar por dependencias funcionales de las columnas de vista materializada.
Funciones agregadas	Las funciones agregadas de consulta deben coincidir o poderse derivar de las funciones agregadas de vista materializada.

**TABLA 16.10**

**Coincidencia de vista materializada y ejemplo 16.20**

	Vista materializada	Consulta
<b>Agrupación</b>	StoreState, TimeYear	StoreState, TimeYear
<b>Condiciones</b>	TimeYear > 2003	TimeYear = 2005 StoreNation = ('USA', 'Canada')
<b>Agregados</b>	SUM(SalesDollar)	SUM(SalesDollar)

- **Coincidencia de agrupación para nivel de detalle:** La reescritura no es posible si la agrupación de la vista materializada tiene un nivel más alto (menos detallado) que una consulta. Desde la perspectiva de una consulta, las columnas de agrupación no deben ser más detalladas que las columnas de agrupación en una vista materializada. Por ejemplo, una consulta con una agrupación en *TimeMonth*, no puede utilizar una vista materializada con una agrupación en *TimeYear*. Sin embargo, puede ascenderse una materializada con agrupación en *TimeMonth* para proporcionar datos para una consulta con agrupación en *TimeYear*.
- **Coincidencia de agrupación para dependencias funcionales:** La reescritura no es posible si una consulta contiene columnas de agrupación que no están en una vista materializada, a menos que las columnas puedan derivarse por dependencias funcionales. Las dependencias funcionales se derivan de llaves primarias, llaves candidatas y dependencias de dimensión (vía la cláusula DETERMINES). Por ejemplo, una consulta con una agrupación en *StoreCity* puede derivarse de una vista materializada con una agrupación en *StoreId*, porque *StoreId* → *StoreCity*. Pueden usarse enlaces para recuperar columnas en una consulta, pero no en una vista materializada en tanto que haya una relación funcional (por lo general una relación 1-M) que implique tablas.
- **Coincidencia de agregado:** Los agregados en la consulta deben coincidir con los agregados disponibles en la vista materializada. Por ejemplo, una consulta que contiene un promedio puede derivarse de una vista materializada que contiene suma y cuenta.

El ejemplo 16.20 presenta un ejemplo de consulta del data warehouse y la consulta reescrita para ilustrar el proceso de coincidencia. La tabla 16.10 ilustra coincidencias entre *MV1* y la consulta del ejemplo 16.20. *MV1* y la consulta coinciden directamente en las columnas de agrupación y los cálculos agregados. La condición en *TimeYear* (> 2003) en *MV1* contiene la condición de consulta (2005). Además, la consulta contiene una condición extra en *StoreNation*. La agrupación no es necesaria en la consulta reescrita porque ya se realizó una agrupación idéntica en *MV1*.

El ejemplo 16.21 presenta un ejemplo más complejo de la reescritura de consultas que implica tres bloques SELECT combinados con el uso del operador UNION. La tabla 16.11 ilustra la coincidencia entre las vistas materializadas (*MV1*, *MV2*, *MV3*) y la consulta del ejemplo 16.21. El primer bloque de la consulta recupera el total de las ventas en Estados Unidos o Canadá, de 2003 a 2006. El segundo bloque de la consulta recupera las ventas por tienda en Estados Unidos

**EJEMPLO 16.20 Consulta de data warehouse y consulta reescrita con el uso de la vista materializada**

```
-- Consulta de data warehouse
SELECT StoreState, TimeYear, SUM(SalesDollar)
FROM Sales, Store, TimeDim
WHERE Sales.StoreId = Store.StoreId
    AND Sales.TimeNo = TimeDim.TimeNo
    AND StoreNation IN ('USA', 'Canada')
    AND TimeYear = 2005
GROUP BY StoreState, TimeYear;

-- Reescritura de Consulta: reemplazar tablas Sales y Time con MV1
SELECT DISTINCT MV1.StoreState, TimeYear, SumDollar1
FROM MV1, Store
WHERE MV1.StoreState = Store.StoreState
    AND TimeYear = 2005
    AND StoreNation IN ('USA','Canada');
```

en 2003. El tercer bloque de la consulta recupera las ventas por tienda en Canadá en 2003. Las cláusulas GROUP BY son necesarias en el segundo y el tercer bloque para ascender al nivel más fino de detalle de las vistas materializadas. En el tercer bloque de la consulta, la condición en *StoreNation* es necesaria porque algunas ciudades tienen nombres idénticos en ambos países.

El ejemplo 16.22 extiende el ejemplo 16.21 con un operador CUBE. En la consulta reescrita, el CUBE se realiza una vez al final, en lugar de hacerlo en cada bloque SELECT. Para ejecutar el CUBE una vez, la cláusula FROM debe contener una consulta empaquetada en un enunciado CREATE VIEW separado.

**EJEMPLO 16.21 Consulta de data warehouse y consulta reescrita con el uso de las vistas materializadas MV1, MV2 y MV3**

```
-- Consulta de data warehouse
SELECT StoreState, TimeYear, SUM(SalesDollar)
FROM Sales, Store, TimeDim
WHERE Sales.StoreId = Store.StoreId
    AND Sales.TimeNo = TimeDim.TimeNo
    AND StoreNation IN ('USA','Canada')
    AND TimeYear BETWEEN 2003 and 2006
GROUP BY StoreState, TimeYear;

-- Reescritura de consulta
SELECT DISTINCT MV1.StoreState, TimeYear, SumDollar1 AS StoreSales
FROM MV1, Store
WHERE MV1.StoreState = Store.StoreState
    AND TimeYear <= 2006
    AND StoreNation IN ('USA','Canada')
UNION
SELECT StoreState, TimeYear, SUM(SumDollar2) as StoreSales
```

```

FROM MV2
WHERE TimeYear = 2003
GROUP BY StoreState, TimeYear
UNION
SELECT DISTINCT StoreState, TimeYear, SUM(SumDollar3) as StoreSales
FROM MV3, Store
WHERE MV3.StoreCity = Store.StoreCity
    AND TimeYear = 2003 AND StoreNation = 'Canada'
GROUP BY StoreState, TimeYear;

```

**TABLA 16.11**  
Coincidencia de vista  
materializadas y  
ejemplo 16.21

	MV1	MV2	MV3	Consulta
<b>Agrupación</b>	StoreState, TimeYear	StoreState, TimeMonth, TimeYear	StoreCity, StoreState, TimeYear	StoreState, TimeYear
<b>Condiciones</b>	TimeYear > 2003	StoreNation = 'USA'	TimeYear <= 2003 StoreNation = 'Canada'	TimeYear BETWEEN 2003 AND 2006 StoreNation = ('USA', 'Canada')
<b>Agregados</b>	SUM(SalesDollar)	SUM(SalesDollar)	SUM(SalesDollar)	SUM(SalesDollar)

Estos ejemplos indican el rango de posibilidades de la reescritura de consultas más que las capacidades de los DBMSs reales. La mayoría de los DBMSs empresariales soportan la reescritura de consultas, pero varía el rango de la reescritura de consultas soportada. Dada la complejidad y la naturaleza patentada de los algoritmos de reescritura de consultas, los detalles de los algoritmos de reescritura de consultas van más allá del alcance de este libro de texto.

#### **EJEMPLO 16.22 Consulta de data warehouse y consulta reescrita con el uso de las vistas materializadas MV1, MV2 y MV3**

```

-- Consulta de data warehouse
SELECT StoreState, TimeYear, SUM(SalesDollar)
    FROM Sales, Store, TimeDim
    WHERE Sales.StoreId = Store.StoreId
        AND Sales.TimeNo = TimeDim.TimeNo
        AND StoreNation IN ('USA', 'Canada')
        AND TimeYear BETWEEN 2003 and 2006
    GROUP BY CUBE(StoreState, TimeYear);

-- Reescritura de consulta
SELECT StoreState, TimeYear, SUM(StoreSales) as SumStoreSales
    FROM (
        SELECT DISTINCT MV1.StoreState, TimeYear, SumDollar1 AS StoreSales
            FROM MV1, Store
            WHERE MV1.StoreState = Store.StoreState
                AND TimeYear <= 2006
                AND StoreNation IN ('USA', 'Canada')
    )

```

```

UNION
SELECT StoreState, TimeYear, SUM(SumDollar2) as StoreSales
  FROM MV2
 WHERE TimeYear = 2003
 GROUP BY StoreState, TimeYear
UNION
SELECT DISTINCT StoreState, TimeYear, SUM(SumDollar3) as StoreSales
  FROM MV3, Store
 WHERE MV3.StoreCity = Store.StoreCity
   AND TimeYear = 2003 AND StoreNation = 'Canada'
 GROUP BY StoreState, TimeYear
 GROUP BY CUBE(StoreState, TimeYear);

```

### 16.3.5 Tecnologías de almacenamiento y optimización

Se han desarrollado varias tecnologías de almacenamiento para ofrecer capacidades de datos multidimensionales. Las tecnologías de almacenamiento soportan On Line Analytic Processing (OLAP), un nombre genérico aplicado a las capacidades de apoyo a las decisiones para los cubos de datos. Esta sección describe las características de las tecnologías de almacenamiento OLAP con un énfasis en las actuales tendencias del mercado.

#### *MOLAP (OLAP Multidimensional)*

En un principio, los fabricantes de software de apoyo a las decisiones desarrollaron una arquitectura de almacenamiento que manipulaba los cubos de datos de manera directa. Esta arquitectura de almacenamiento, conocida como MOLAP, por OLAP Multidimensional, era la única opción como tecnología de almacenamiento para los data warehouse hasta mediados de la década de los noventa. En la actualidad, MOLAP se ha visto eclipsado como la principal arquitectura de almacenamiento para los data warehouse, pero sigue siendo una tecnología importante para los cubos de datos de resumen y pequeños data warehouse y data mart.

Los mecanismos de almacenamiento MOLAP manipulan directamente los cubos de datos almacenados. Los mecanismos de almacenamiento de los sistemas MOLAP se optimizan para las características únicas de los datos multidimensionales, como esparcimiento y agregación compleja en miles de celdas. Como los cubos de datos se precisan, el desempeño de la consulta MOLAP por lo general es mejor que los planteamientos competidores que usan el almacenamiento de base de datos relacional. Aun con técnicas para manejar el esparcimiento, los mecanismos de MOLAP pueden verse abrumados por el tamaño de los cubos de datos. Un cubo de datos calculado por completo puede expandirse muchas veces en comparación con los datos de entrada en bruto. Este problema de explosión de datos limita el tamaño de los cubos de datos que los mecanismos MOLAP pueden manipular.

#### *ROLAP (OLAP Relacional)*

Gracias al tamaño del mercado potencial y al crecimiento del procesamiento de data warehouse, los fabricantes de DBMS relacionales han extendido sus productos con características adicionales para soportar operaciones y estructuras de almacenamiento para datos multidimensionales. Estas extensiones de los productos se conocen colectivamente como ROLAP, por OLAP Relacional. Debido al tamaño creciente de los data warehouse y a su investigación y desarrollo intensivos por parte de fabricantes de DBMS, fue sólo cuestión de tiempo para que el ROLAP dominara el mecanismo de almacenamiento para los data warehouse.

En el planteamiento del ROLAP, las bases de datos relacionales almacenan datos multidimensionales con el uso del esquema de estrella o sus variaciones, como se describió en la sección 16.3.1. Los cubos de datos se construyen dinámicamente a partir de tablas de hechos y

#### **MOLAP**

mecanismo de almacenamiento que almacena y manipula directamente cubos de datos. Los mecanismos MOLAP generalmente ofrecen el mejor desempeño de la consulta, pero ponen límites para el tamaño de los cubos de datos.

#### **ROLAP**

extensiones del DBMS relacional para soportar datos multidimensionales. Los mecanismos ROLAP soportan una variedad de técnicas de almacenamiento y optimización para la recuperación de datos de resumen.

dimensiones, así como de vistas materializadas. Por lo regular, sólo debe construirse un subconjunto de datos, como lo especifica la consulta de un usuario. Las extensiones para SQL permiten a los usuarios manipular las dimensiones y medidas en cubos de datos virtuales, como se describió en la sección 16.3.3.

Los mecanismos ROLAP incorporan una variedad de técnicas de almacenamiento y optimización para la recuperación de datos de resumen. Esta lista explica las técnicas más sobresalientes.

- **Índices de enlace de mapa de bits** (véase el capítulo 8 para más detalles) son particularmente útiles para las columnas en las tablas de dimensión con unos cuantos valores, como *CustState*. Un índice de enlace de mapa de bits proporciona un enlace precalculado de los valores de una tabla de dimensión para las filas de una tabla de hechos. Para soportar los esquemas de copo de nieve, algunos fabricantes de DBMS soportan índices de enlace de mapa de bits para tablas de dimensión relacionadas con otras tablas de dimensión. Por ejemplo, un índice de mapa de bits para la columna *Division.DivManager* tiene un registro de índice que contiene un mapa de bits para filas relacionadas de la tabla *Store*, y un segundo mapa de bits para filas de la tabla *Sales* relacionadas con las filas de coincidencia de la tabla *Store*.
- **Optimización de consultas con enlace de estrella** utiliza índices de enlace de mapa de bits en tablas de dimensiones para reducir el número de filas en la tabla de hechos que debe recuperarse. Un enlace de estrella implica una tabla de hechos enlazada con una o más tablas de dimensión. La optimización de enlace de estrella incluye tres fases. En la primera fase, los índices de enlace de mapa de bits en cada tabla de dimensión se combinan usando el operador de unión para condiciones conectadas por el operador OR, y el operador de intersección para condiciones conectadas por el operador AND. En la segunda fase, los mapas de bits que resultan de la primera fase se combinan utilizando el operador de intersección. En la tercera fase, las filas de la tabla de hechos se recuperan con el uso del mapa de bits resultante de la segunda fase. La optimización de enlace de estrella puede dar como resultado un tiempo de ejecución sustancialmente menor en comparación con los algoritmos de enlace tradicionales que combinan dos tablas al mismo tiempo.
- **Reescritura de consultas usando vistas materializadas** pueden eliminar la necesidad de tener acceso a grandes tablas de hechos y dimensión. Si las vistas materializadas son grandes, pueden indexarse para mejorar el desempeño de la recuperación. La reescritura de consultas utiliza el optimizador de consultas para evaluar el beneficio de emplear vistas materializadas en comparación con las tablas de hechos y dimensión.
- **Asesores de almacenamiento de resumen** determinan el mejor conjunto de vistas materializadas que debería crearse y mantenerse para una carga de trabajo de consulta dada. Para que haya consistencia con otros componentes, el asesor de resumen se integra con el componente de reescritura de consultas y el optimizador de consultas.
- **Partición, simplificación y ejecución paralela de consultas** ofrecen oportunidades para disminuir el tiempo de ejecución de las consultas al data warehouse. Es necesario estudiar con cuidado las elecciones, de modo que el uso de la partición y simplificación soporten el nivel deseado de ejecución paralela de consultas.

## HOLAP

mecanismo de almacenamiento para data warehouse que combina mecanismos de almacenamiento ROLAP y MOLAP. El HOLAP implica tanto almacenamiento de datos relacionales y multidimensionales, como combinación de datos de fuentes relacionales y multidimensionales para operaciones de cubo de datos.

A pesar de la investigación y el desarrollo intensivos sobre técnicas de almacenamiento y optimización ROLAP, los mecanismos MOLAP darán un mejor tiempo de respuesta a las consultas. No obstante, el almacenamiento MOLAP sufre de limitaciones en el tamaño del cubo de datos, de manera que el almacenamiento ROLAP es preferible para data warehouse detallados. Además, la diferencia en el tiempo de respuesta se ha estrechado tanto que el almacenamiento ROLAP puede implicar sólo una ligera desventaja en el desempeño si se emplean en forma adecuada las técnicas de almacenamiento y optimización ROLAP.

## *HOLAP (OLAP híbrido)*

Debido al equilibrio entre MOLAP y ROLAP, se ha desarrollado una tercera tecnología conocida como **HOLAP**, por OLAP híbrido, para combinar ROLAP y MOLAP. El HOLAP permite que un data warehouse se divida entre el almacenamiento relacional de tablas de hechos y

dimensiones, y el almacenamiento multidimensional de cubos de datos de resumen. Cuando se presenta una consulta OLAP, el sistema HOLAP puede combinar los datos administrados por ROLAP y MOLAP.

A pesar del atractivo del HOLAP, tiene desventajas potenciales que pueden limitar su uso. Primero, HOLAP puede ser más complejo que ROLAP o MOLAP, en especial si un fabricante de DBMS no ofrece soporte total para HOLAP. Para poder soportar por completo HOLAP, un fabricante de DBMS debe proporcionar mecanismos tanto MOLAP como ROLAP, así como herramientas para combinar ambos mecanismos en el diseño y operación de un data warehouse. Segundo, hay una transposición considerable en la funcionalidad entre las técnicas de almacenamiento y la optimización en los mecanismos ROLAP y MOLAP. No está claro si las técnicas de almacenamiento y optimización ROLAP deben eliminarse o usarse además de las técnicas MOLAP. Tercero, debido a que la diferencia en el tiempo de respuesta se ha reducido entre ROLAP y MOLAP, su combinación quizás no ofrezca una mejora considerable en el desempeño como para justificar la complejidad mayor.

## 16.4 Mantenimiento de un data warehouse

---

Aunque los data warehouse contienen en gran medida datos reproducidos, el mantenimiento de un data warehouse es mucho más difícil que sólo copiar desde los datos de origen. Mantener un data warehouse implica inicialmente poblar el almacén con datos de origen y refrescarlo en forma periódica conforme cambian los datos de origen. La determinación para cargar en un almacén implica hacer que las necesidades de apoyo a las decisiones coincidan con las realidades de los datos disponibles. Para satisfacer las necesidades de apoyo a las decisiones, los data warehouse usan datos de muchas fuentes, tanto internas como externas. La reconciliación de las diferencias entre las fuentes de datos es un desafío significativo, en especial si consideramos que por lo general los sistemas de origen no pueden cambiarse. Conforme cambian sus datos de origen, el data warehouse debe refrescarse de modo oportuno para soportar la toma de decisiones. La determinación del tiempo y contenido que se debe refrescar puede ser un reto importante debido a que las fuentes de datos cambian con distintos índices. Como consecuencia de estos desafíos, el mantenimiento de un data warehouse es tal vez la actividad de mayor importancia en el desarrollo de un data warehouse.

Esta sección presenta varios aspectos importantes del mantenimiento de un data warehouse. La primera parte describe las clases de datos de origen disponibles para poblar un data warehouse. La segunda parte describe el flujo de trabajo para mantener un almacén, y la parte final analiza el problema que representa determinar la frecuencia del refrescamiento y el contenido que se debe refrescar.

### 16.4.1 Fuentes de datos

El acceso a los datos de origen presenta desafíos en el manejo de una variedad de formatos y restricciones de los sistemas de origen. Por lo regular, no es posible cambiar los sistemas de origen externo. Los sistemas de origen interno pueden o no cambiarse para ajustarse a los requerimientos de un data warehouse. Aun cuando pueda cambiarse un sistema de origen, es probable que las limitaciones presupuestales sólo permitan cambios menores. Los datos de origen pueden almacenarse en formato heredado o formato moderno. El formato heredado generalmente evita la recuperación usando lenguajes no procedurales como SQL. El formato moderno significa que puede tenerse acceso a los datos de origen como una base de datos relacional o como páginas web. Las páginas web pueden ser difíciles de aparecerse y no podrán estandarizarse de un sitio web a otro a menos que se almacenen con metadatos formales.

El cambio de datos desde los sistemas de origen proporciona la base para actualizar un data warehouse. El cambio de datos compromete el nuevo origen de datos (inserciones) y modificaciones al origen de datos existente (actualizaciones o eliminaciones). Además, el cambio de datos puede afectar tablas de hechos y/o tablas de dimensiones. Los cambios de datos más comunes implican inserciones de hechos nuevos. Las inserciones de nuevas dimensiones y modificaciones de dimensiones son menos comunes, pero su captura sigue siendo importante.

**TABLA 16.12**  
**Resumen de la clasificación del cambio de datos**

Tipo de cambio	Descripción	Evaluación
Cooperativo	Notificación del sistema de origen con el uso de disparadores	Requiere modificaciones de los sistemas de origen
Registrado	Actividad del sistema de origen capturada en registros	Fácilmente disponibles pero con procesamiento significativo para extraer datos útiles
Consultable	Consultas del sistema de origen usando estampas de tiempo	Requiere estampas de tiempo en los datos de origen y sistemas de origen no heredados
Fotografía instantánea	Descarga repentina de datos de origen aumentados con operaciones de diferencia	Uso intensivo de recursos para su creación y procesamiento significativo para operaciones de diferencia; ningún requerimiento de sistema de origen tan útil para los datos heredados

De acuerdo con las características del sistema de origen, los datos de cambio pueden clasificarse como de cooperación, registrados, consultables o de fotografía instantánea, como se resume en la tabla 16.12. El cambio de datos cooperativo implica la notificación del sistema de origen acerca de los cambios. La notificación por lo general ocurre con el uso de un disparador en el momento en que termina la transacción. El cambio de datos cooperativo puede capturarse de inmediato en un data warehouse o ponerse en una cola para su posible entrada posterior con otros cambios. El cambio de datos cooperativo es el formato menos común para el cambio de datos porque implica modificaciones tanto al sistema de origen como al data warehouse.

El cambio de datos registrados implica archivos que registran cambios u otra actividad del usuario. Por ejemplo, un registro de transacción contiene todos los cambios realizados por una transacción y un registro web contiene históricos de acceso a páginas por parte de los visitantes de sitios web (llamado flujo de clics). El cambio de datos registrados normalmente no implica cambios en un sistema de origen, puesto que ya están disponibles como registros para la mayor parte de los recursos del sistema. Sin embargo, los registros pueden contener grandes cantidades de datos irrelevantes. Además, la derivación de los datos relevantes necesarios para el data warehouse puede requerir la comparación del registro de sucesos relacionados, una tarea de recursos intensivos. El cambio de datos registrados se emplea con mayor frecuencia para el subconjunto de relaciones con los clientes de un data warehouse.

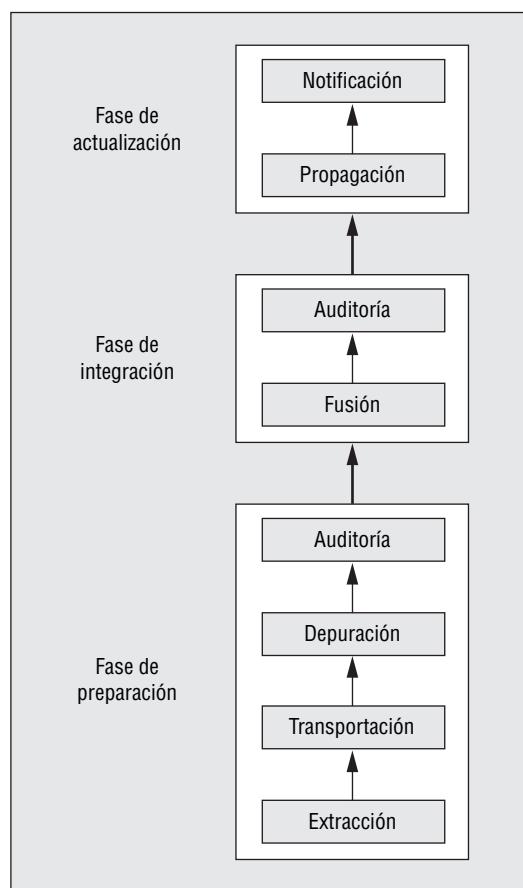
Como su nombre lo indica, el cambio de datos consultables proviene de una fuente de datos a través de una consulta. El cambio de datos consultables requiere del estampado de tiempo en la fuente de datos. Debido a que unas cuantas fuentes de datos contienen estampas de tiempo para todos los datos, el cambio de datos consultables por lo regular se aumenta con otra clase de datos. El cambio de datos consultables aplica más en tablas de hechos con el uso de campos como fecha de pedido, fecha de embarque y fecha de contratación, que se almacenan como fuente de datos operativos.

El cambio de datos de fotografía instantánea implica descargas repentinas periódicas de los datos de origen. Para derivar los datos de cambio, una operación de diferencia utiliza las dos fotografías instantáneas más recientes. El resultado de una operación de diferencia se conoce como delta. Las fotografías instantáneas son la forma más común de cambio de datos como consecuencia de la aplicabilidad. Las fotografías instantáneas son la única forma de cambio de datos sin requerimientos en un sistema de origen. Calcular una fotografía instantánea puede ocupar recursos intensivos, por lo que puede haber restricciones acerca del tiempo y la frecuencia de recuperación de una fotografía instantánea.

#### 16.4.2 Flujo de trabajo para el mantenimiento de un data warehouse

Mantener un data warehouse implica una variedad de tareas que manipulan el cambio de datos de los sistemas de origen. La figura 16.15 presenta un flujo de trabajo genérico que organiza las

**FIGURA 16.15**  
Flujo de trabajo genérico para el mantenimiento del data warehouse



tareas. La fase de preparación manipula el cambio de datos de sistemas de origen individual. La extracción implica la recuperación de datos de un sistema de origen individual. La transportación implica el movimiento de los datos extraídos a un área de estancia. La depuración implica una variedad de tareas para estandarizar y mejorar la calidad de los datos extraídos. La auditoría implica registrar los resultados del proceso de depuración, hacer revisiones sobre la integridad y racionalidad y manejar excepciones.

La fase de integración implica fusionar las fuentes separadas y depuradas. La fusión puede implicar la eliminación de inconsistencias entre los datos de origen. La auditoría implica registrar los resultados del proceso de fusión, efectuar revisiones de integridad y racionalidad y manejar excepciones.

La fase de actualización implica propagar el cambio de datos integrados a varias partes del data warehouse, incluidas tablas de hechos y dimensiones, vistas materializadas, cubos de datos almacenados y data marts. Después de la propagación, puede enviarse una notificación a los grupos de usuarios y administradores.

Las fases de preparación e integración deberían resolver los problemas de calidad de los datos, como se resume en la tabla 16.13. Los datos de sistemas heredados normalmente son sucios, lo que significa que pueden no conformar los estándares de calidad de datos entre ellos o entre los datos de toda la empresa. Los datos sucios pueden llevar a una toma de decisiones deficientes si se cargan directamente. La auditoría debe incluir el manejo de excepciones con el fin de resolver los problemas de calidad de los datos. Las excepciones pueden anotarse en un archivo de registro y luego manejarse manualmente. Con el paso del tiempo, las excepciones tendrían que disminuir conforme se mejoran los estándares de calidad de los datos en las fuentes de datos internas.

Además del manejo de excepciones, la tarea de auditoría debe incluir revisiones de la integridad y la racionalidad. Una revisión de la integridad cuenta el número de unidades de reporte

**TABLA 16.13**

**Problemas comunes en la calidad de los datos**

<b>Identificadores múltiples:</b> algunas fuentes de datos pueden utilizar diferentes llaves primarias para la misma entidad como números de cliente diferentes
<b>Nombres múltiples:</b> el mismo campo puede representarse usando nombres de campo distintos
<b>Diferentes unidades:</b> las medidas y dimensiones pueden tener diferentes unidades y granularidades
<b>Valores faltantes:</b> es probable que los datos no existan en algunas bases de datos; pueden utilizarse diversos valores predeterminados en las fuentes de datos para compensar los valores faltantes
<b>Transacciones huérfanas:</b> algunas transacciones pueden ser partes importantes perdidas, como un pedido sin cliente
<b>Campos multipropósito:</b> algunas bases de datos pueden combinar datos en un campo, como diferentes componentes de una dirección
<b>Datos en conflicto:</b> algunas fuentes de datos pueden tener datos en conflicto, como diferentes domicilios de un cliente
<b>Diferentes horas de actualización:</b> algunas fuentes de datos pueden hacer actualizaciones en diferentes intervalos

para asegurarse de que todas hayan reportado durante un periodo dado. Una revisión de la razonabilidad determina si los hechos clave caen dentro de límites predeterminados y son una extrapolación realista de la historia previa. Las excepciones pueden requerir reconciliación por parte de los analistas de la empresa antes de la propagación al data warehouse.

El flujo de trabajo genérico de la figura 16.15 se aplica tanto a la carga inicial como al refrescamiento periódico de un almacén. La carga inicial a menudo requiere un periodo extenso de depuración de datos o resolver los problemas de la calidad de los mismos. Un objetivo de la carga inicial es descubrir los problemas de calidad de los datos y solucionarlos. El proceso de refrescamiento comúnmente varía entre fuentes de datos. El proceso del flujo de trabajo debe personalizarse para ajustarse a los requerimientos de cada fuente de datos. Por ejemplo, puede minimizarse la auditoría para las fuentes de datos de alta calidad.

Para soportar la complejidad del mantenimiento del data warehouse, se han desarrollado productos de software que reciben el nombre de herramientas de extracción, transformación y carga (ETL). Estas herramientas, disponibles de terceros fabricantes y fabricantes de DBMS, eliminan la necesidad de escribir código personalizado para muchas tareas de mantenimiento. La mayor parte de las herramientas de ETL utilizan especificaciones de reglas en vez de codificación procedural para indicar la lógica y las acciones. Algunas herramientas de ETL pueden generar código que puede personalizarse para mayor flexibilidad. Además de las herramientas de ETL, algunos fabricantes de DBMS proporcionan programas de carga de datos y extensiones SQL patentadas que soportan las tareas de mantenimiento. Por ejemplo, Oracle 10g soporta el programa SQL Loader, así como el enunciado MERGE y el enunciado de tablas múltiples INSERT. Las herramientas de ETL y extensiones SQL patentadas son esenciales para reducir el esfuerzo de implementar las tareas del flujo de trabajo.

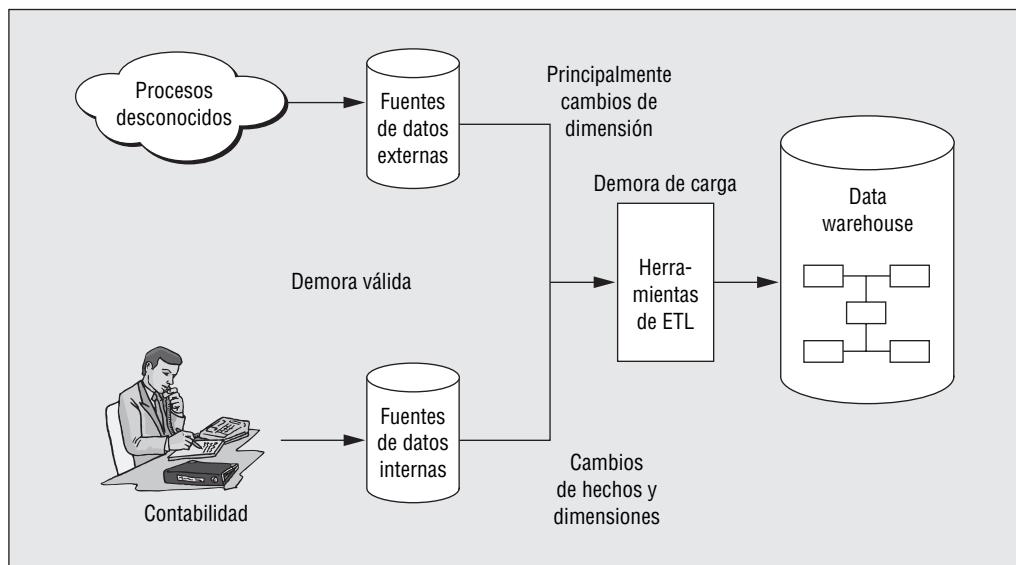
### 16.4.3 Administración del proceso de refrescamiento

Refrescar un data warehouse es un proceso complejo que implica la administración de diferencias de hora entre la actualización de fuentes de datos y la actualización de los objetos relacionados en el data warehouse (tablas, vistas materializadas, cubos de datos, data marts). En la figura 16.16, la demora válida es la diferencia entre el acontecimiento de un evento en el mundo real (hora válida) y el almacenamiento del evento en una base de datos operacional (hora de la transacción). La demora de carga es la diferencia entre la hora de la transacción y el almacenamiento del evento en un almacén (tiempo de carga). Para fuentes de datos internas, puede haber cierto control sobre la demora válida. Por lo general, no se tiene control sobre la demora válida en el caso de las fuentes de datos externas. Por consiguiente, un administrador de data warehouse debe tener control sobre la demora de carga.

La figura 16.13 implica que el origen de datos puede cambiarse de manera independiente, llevándolo a diferentes índices de cambio para las tablas de hechos y tablas de dimensiones. Las tablas de hechos generalmente registran eventos concluidos, como pedidos, embarques y compras con vínculos a dimensiones relacionadas. Por ejemplo, insertar una fila en la tabla *Sales* requiere

**herramientas ETL**  
herramientas de software para extracción, transformación y carga de datos desde las fuentes de datos hasta el data warehouse. Las herramientas ETL eliminan la necesidad de escribir código personalizado para muchas tareas de mantenimiento del data warehouse.

**FIGURA 16.16**  
Panorama del proceso de refrescamiento del data warehouse



llaves foráneas que hacen referencia a las tablas *Customer*, *Store*, *TimeDim* e *Item*. Sin embargo, las actualizaciones e inserciones a las tablas de dimensión relacionadas pueden ocurrir a diferentes horas que los eventos de hechos. Por ejemplo, un cliente puede moverse o un artículo puede cambiar su precio en diferentes horas que las de los pedidos, embarques y compras de inventario. Como resultado de los diferentes índices de cambio, un administrador de data warehouse debe administrar la demora de carga por separado para las tablas de dimensiones y de hechos.

El principal objetivo de la administración del proceso de refrescamiento es determinar la frecuencia del refrescamiento para cada fuente de datos. La frecuencia de refrescamiento óptima maximiza el beneficio de refrescamiento neto, definido como el valor de las líneas de tiempo de los datos menos el costo del refrescamiento. El valor de la línea de tiempo de los datos depende de la sensibilidad de la toma de decisiones para la vigencia de los datos. Algunas decisiones son muy sensibles al tiempo, como las decisiones de inventario. Las organizaciones de una cadena de abastecimiento intentan minimizar los costos implícitos de inventario al almacenar bienes tan cerca como sea posible al momento de necesitarlos. Otras decisiones no son tan sensibles al tiempo. Por ejemplo, la decisión de cerrar una tienda con desempeño deficiente por lo regular se tomaría usando datos de un extenso periodo.

Fisher y Berndt (2001) han propuesto un método para medir el valor de la línea de tiempo de los datos. Definieron una medida del índice de error de un data warehouse con respecto a una carga de trabajo de consultas dada. Para determinar el índice de error, se estima la volatilidad de los datos de las consultas y dimensiones. Aun cuando un data warehouse almacena datos de nivel individual, la mayoría de las consultas implicarán adiciones, no recuperación de datos de nivel individual. Por ejemplo, la mayor parte de las consultas en el esquema de ventas del almacén implicarían la marca del artículo o su categoría, no los artículos individuales. Dada una evaluación del índice de error, puede tenerse acceso al valor de la línea de tiempo de los datos usando la frecuencia u otra ponderación para las consultas al data warehouse.

El costo de refrescar un data warehouse incluye tanto recursos de computación como humanos. Los recursos de computación son necesarios para todas las tareas en el flujo de trabajo del mantenimiento. Los recursos humanos pueden ser necesarios en las tareas de auditoría durante las fases de preparación e integración. El nivel de la calidad de los datos en los datos de origen también afecta el nivel de los recursos humanos requeridos. El esfuerzo de desarrollo para usar herramientas de ETL y escribir software personalizado no es parte del costo de refrescamiento, a menos que haya un costo de desarrollo en curso con cada refrescamiento. Una distinción importante implica el costo fijo y el costo variable del refrescamiento. Un alto costo fijo alienta el refrescamiento menos frecuente, porque el costo fijo ocurre con cada refrescamiento. El costo fijo puede incluir un esfuerzo de inicio y cierre, así como renta de recursos.

**TABLA 16.14**  
**Resumen de las limitaciones de refreshamiento**

Tipo de limitación	Descripción
Acceso de origen	Restricciones en el tiempo y frecuencia de la extracción de cambio de datos
Integración	Restricciones que requieren reconciliación actual del cambio de datos
Integridad/consistencia	Restricciones que requieren carga de datos de cambio en el mismo periodo de refrescamiento
Disponibilidad	Restricciones de programación de carga como resultado de los aspectos de recursos incluida capacidad de almacenamiento, disponibilidad en línea y uso del servidor

El administrador del data warehouse debe satisfacer las restricciones sobre el proceso de refrescamiento junto con la cuadratura del valor de la línea de tiempo contra el costo del refrescamiento, como se resume en la tabla 16.14. Las restricciones, ya sea sobre un data warehouse o un sistema de origen, pueden limitar el refrescamiento frecuente. Las restricciones de acceso a la fuente pueden ser consecuencia de una tecnología heredada con escalabilidad restringida para fuentes de datos internas o problemas de coordinación para fuentes de datos externas. Las restricciones de integración a menudo implican identificación de entidades comunes como clientes y transacciones entre fuentes de datos. Las restricciones de integridad/consistencia pueden implicar mantenimiento del mismo periodo de tiempo en el cambio de datos o la inclusión del cambio de datos de cada fuente de datos para la integridad. La disponibilidad del data warehouse con frecuencia implica conflictos entre la disponibilidad en línea y la carga del data warehouse.

## Reflexión final

Este capítulo presentó una introducción a los conceptos y práctica del almacenamiento de datos. Este capítulo empezó por examinar las diferencias conceptuales entre bases de datos relacionales, empleadas tradicionalmente para el procesamiento de transacciones, y bases de datos multidimensionales, sugeridas para la nueva generación de aplicaciones de apoyo a las decisiones. Se describieron las características únicas de datos de apoyo a las decisiones, seguido de un análisis de las arquitecturas de data warehouse, minería de datos y uso del data warehouse en las organizaciones.

Puede implementarse data warehouse usando el modelo de datos multidimensionales, el modelo relacional o una combinación de ambos modelos. Para el modelo de datos multidimensionales, este capítulo presentó la terminología asociada con los cubos de datos y operadores para manipular cubos de datos. Para el modelo relacional de datos, el capítulo presentó técnicas de modelado de datos (el esquema de estrella y sus variaciones), extensiones SQL en la cláusula GROUP BY para los datos dimensionales de consulta, vistas materializadas y tecnologías de almacenamiento. Las técnicas de almacenamiento soportan data warehouse con el uso tanto de mecanismos de relación como de almacenamiento de cubos de datos.

No obstante el modelo de datos y la arquitectura de almacenamiento, mantener un data warehouse es un proceso difícil que debe manejarse con cuidado. El capítulo presentó los tipos de fuentes de datos usadas en el mantenimiento de un data warehouse, un flujo de trabajo genérico que describe las tareas involucradas en el mantenimiento de un data warehouse, así como aspectos de diseño a considerar en el proceso de refrescamiento. El capítulo propuso el uso de las herramientas de ETL para reducir la cantidad de código de personalización para implementar los procedimientos que pueblan un data warehouse.

## Revisión de conceptos

- Necesidades de datos para el procesamiento de transacciones contra aplicaciones de apoyo a las decisiones.
- Características de un data warehouse: orientado a los sujetos, integrado, variable con el tiempo y no volátil.
- Arquitecturas para desplegar un data warehouse: dos niveles, tres niveles y ascendente.

- Etapas del desarrollo de un data warehouse (prenatal, infancia, niñez, adolescencia, edad adulta y sabiduría) y dificultad de moverse entre etapas (infancia a niñez y adolescencia a edad adulta).
- Cubo de datos multidimensionales: dimensiones, medidas, jerarquías, tipo de datos de serie de tiempo.
- Operadores multidimensionales: rebanada, dado, descenso, ascenso, pivote.
- Esquema de estrella: tabla de hechos y tablas de dimensiones relacionadas.
- Variaciones del esquema de estrella: esquema de copo de nieve (niveles de dimensión múltiple) y esquema de constelación (múltiples tablas de hechos).
- Mantenimiento de la integridad dimensional histórica con el uso de una representación tipo II para historial ilimitado y una representación tipo III para historial limitado.
- Representación de dimensiones para soportar las operaciones de cubo de datos y las técnicas de reescritura de consultas.
- Extensiones de la cláusula GROUP BY para cálculo de subtotales: operadores CUBE, ROLLUP y GROUPING SETS.
- Vistas materializadas para almacenamiento de datos de resumen precalculados.
- Reescritura de consultas que implican sustitución de vistas materializadas para tablas de hechos y dimensiones con el fin de mejorar el desempeño de las consultas al data warehouse.
- Arquitectura de almacenamiento multidimensional: ROLAP, MOLAP y HOLAP.
- Tipos de cambio de datos usados para poblar un data warehouse: cooperativo, registrado, consultables y de fotografía instantánea.
- Fases del flujo de trabajo para mantener un almacén: preparación, integración y propagación.
- Importancia de las herramientas de ETL (extracción, transformación y carga) para reducir la codificación en los procedimientos para poblar un data warehouse.
- Determinación de la frecuencia de refrescamiento para un data warehouse: equilibrio de la frecuencia del refrescamiento contra el costo del refrescamiento mientras se satisfacen las restricciones de refrescamiento.
- Tipos de restricciones de refrescamiento: acceso a fuente, integración, integridad/consistencia, disponibilidad.

## Preguntas

1. ¿Por qué las bases de datos operacionales no son particularmente apropiadas para las aplicaciones de apoyo a las decisiones?
2. ¿En qué difiere un data warehouse de un mercado de datos?
3. ¿Cuándo es más adecuada la arquitectura de data warehouse de tres niveles que la de dos niveles?
4. ¿Cuáles son los componentes de un modelo de datos empresariales?
5. ¿Cuáles son algunas causas de los fracasos en los proyectos de data warehouse?
6. ¿Una arquitectura ascendente de data warehouse utiliza un modelo de datos empresariales?
7. ¿Qué es un mercado operativo?
8. ¿Cuál es el propósito del modelo de madurez del data warehouse?
9. ¿Qué discernimiento importante ofrece el modelo de madurez del data warehouse?
10. ¿Cuáles son las ventajas de la representación multidimensional sobre la representación relacional para los analistas empresariales?
11. Explique por qué una dimensión puede tener múltiples jerarquías.
12. ¿Cuáles son las ventajas del uso de datos de serie de tiempo en una celda en vez del tiempo como una dimensión?
13. ¿En qué difiere el rebanado de un cubo de datos del manejo de datos?
14. ¿Cuáles son las diferencias entre el descenso y el ascenso de una dimensión de cubo de datos?

15. ¿Cómo sirve una operación de pivote a las bases de datos multidimensionales?
16. Explique la importancia del esparcimiento en un cubo de datos.
17. ¿Qué es un esquema de estrella?
18. ¿Cuáles son las diferencias entre las tablas de hechos y las tablas de dimensiones?
19. ¿En qué difiere un esquema de copo de nieve de un esquema de estrella?
20. ¿Qué es un esquema de constelación?
21. ¿Cómo se representa el tiempo para una tabla de hechos?
22. ¿Qué es una tabla de hechos acumulativos?
23. ¿Cuál es la diferencia entre las representaciones de tipo II y tipo III para la integridad dimensional histórica?
24. ¿Cuál es el propósito del enunciado de Oracle CREATE DIMENSION?
25. ¿Cuál es el propósito del operador CUBE de SQL?
26. ¿Cuál es el propósito del operador ROLLUP de SQL?
27. ¿Cuál es el propósito del operador GROUPING SETS de SQL?
28. Liste brevemente algunas de las variaciones de los operadores CUBE, ROLLUP y GROUPING SETS.
29. ¿Por qué son importantes las vistas materializadas para los data warehouse, pero no lo son para las bases de datos operacionales?
30. ¿Qué propiedades de materialización ofrece Oracle 10g para las vistas materializadas?
31. Compare y contraste la reescritura de consultas para las vistas materializadas con la modificación de consultas para las vistas tradicionales (no materializadas).
32. Explique brevemente los procesos de coincidencia para permitir la reescritura de consultas.
33. Explique la importancia de la indexación de tablas de hechos y dimensiones en un data warehouse.
34. ¿Cuáles son las ventajas y desventajas del mecanismo de almacenamiento MOLAP?
35. ¿Cuáles son las ventajas y desventajas del mecanismo de almacenamiento ROLAP?
36. ¿Cuáles son las ventajas y desventajas del mecanismo de almacenamiento HOLAP?
37. Mencione algunas técnicas de almacenamiento y optimización en mecanismos ROLAP.
38. ¿Qué es un cambio de dato cooperativo?
39. ¿Qué es un cambio de dato registrado?
40. ¿Qué es un cambio de dato consultable?
41. ¿Qué es un cambio de dato de fotografía instantánea?
42. Describa brevemente las fases de mantenimiento del data warehouse.
43. Mencione problemas comunes en la calidad de los datos que se deben resolver mediante las fases de preparación e integración.
44. ¿Cuál es el beneficio de usar herramientas de ETL en el mantenimiento del data warehouse?
45. ¿Cuál es la demora válida?
46. ¿Cuál es la demora de tiempo de carga?
47. ¿Cuál es el principal objetivo de la administración del proceso de refrescamiento para un data warehouse?
48. ¿Qué tipos de restricciones afectan el proceso de refrescamiento?

## Problemas

### Parte 1: Problemas de seguro de automóvil

La parte 1 le permite practicar con la definición y las operaciones de cubos de datos, así como con esquemas de estrella. Las preguntas en la parte 1 comprenden la base de datos que mostramos a continuación y que utiliza un proveedor de seguros de automóvil para soportar las transacciones de las pólizas (crear y mantener las pólizas de los clientes) y las transacciones de quejas (quejas presentadas por otras partes). Las transacciones de pólizas utilizan las tablas *Item*, *Agent*, *InsuredParty*, *Policy*, *InsuredAuto* y *PolicyItem*, mientras que las transacciones de quejas usan las tablas *InsuredParty*, *Claimant*, *ThirdParty*, *InsuredAuto*, *Policy* y *Claim*. Se muestran las nueve tablas en esta base de datos, así como las llaves primarias (en negritas) y las llaves foráneas (en cursivas). Para cada tabla, suponga los formatos de los campos a su elección.

**Tabla: Item**

<b>ItemNo</b>	ItemDesc	ItemMinCoverage	ItemMaxCoverage
---------------	----------	-----------------	-----------------

Llave primaria: ItemNo  
 Llaves foráneas: Ninguna

**Tabla: Agent**

<b>AgentNo</b>	AgentName	AgentPhone	AgentDept	AgentType	AgentRegion
----------------	-----------	------------	-----------	-----------	-------------

Llave primaria: AgentNo  
 Llaves foráneas: Ninguna

**Tabla: InsuredParty**

<b>IPSSN</b>	IPDrvLicNo	IPState	IPName	IPPhone	IPAddr	IPDOB	IPCity	IPZip	IPRiskCat
--------------	------------	---------	--------	---------	--------	-------	--------	-------	-----------

Llave primaria: IPSSN  
 Llaves foráneas: Ninguna  
 IPDOB es la fecha de nacimiento de la parte asegurada

**Tabla: Claimant**

<b>ClmtNo</b>	ClmtName	ClmtPhone	ClmtInsComp	ClmtPolNo	ClmtAddr	ClmtCity	ClmtState	ClmtZip
---------------	----------	-----------	-------------	-----------	----------	----------	-----------	---------

Llave primaria: ClmtNo  
 Llaves foráneas: Ninguna

**Tabla: ThirdParty**

<b>TPSSN</b>	TPName	TPPhone	TPDesc	TPAddr	TPCity	TPState	TPZip
--------------	--------	---------	--------	--------	--------	---------	-------

Llave primaria: TPSSN  
 Llaves foráneas: Ninguna

**Tabla: Policy**

<b>PolNo</b>	PolBegDate	PolEndDate	PSSN	AgentNo	PolPremium	PolEffDate
--------------	------------	------------	------	---------	------------	------------

Llave primaria: PolNo  
 Llaves foráneas: IPSSN, AgentNo

**Tabla: InsuredAuto**

<b>IAVIN</b>	IALicPlateNo	IAState	IAMake	IAModel	IAYear	IAAirBags	IADriverSSN	PolNo
--------------	--------------	---------	--------	---------	--------	-----------	-------------	-------

Llave primaria: IPVIN  
 Llaves foráneas: IADriverSSN (se refiere a InsuredParty), PolNo

**Tabla: PolicyItem**

<b>IAVIN</b>	<b>ItemNo</b>	PICoverage	PIPremium	PIComments
--------------	---------------	------------	-----------	------------

Llave primaria: IAVIN, ItemNo  
 Llaves foráneas: IAVIN, ItemNo

**Tabla: Claim**

<b>ClaimNo</b>	ClaimAmount	ClaimEstimate	ClaimDesc	ClaimDate	IAVIN	ClmtNo	TPSSN
----------------	-------------	---------------	-----------	-----------	-------	--------	-------

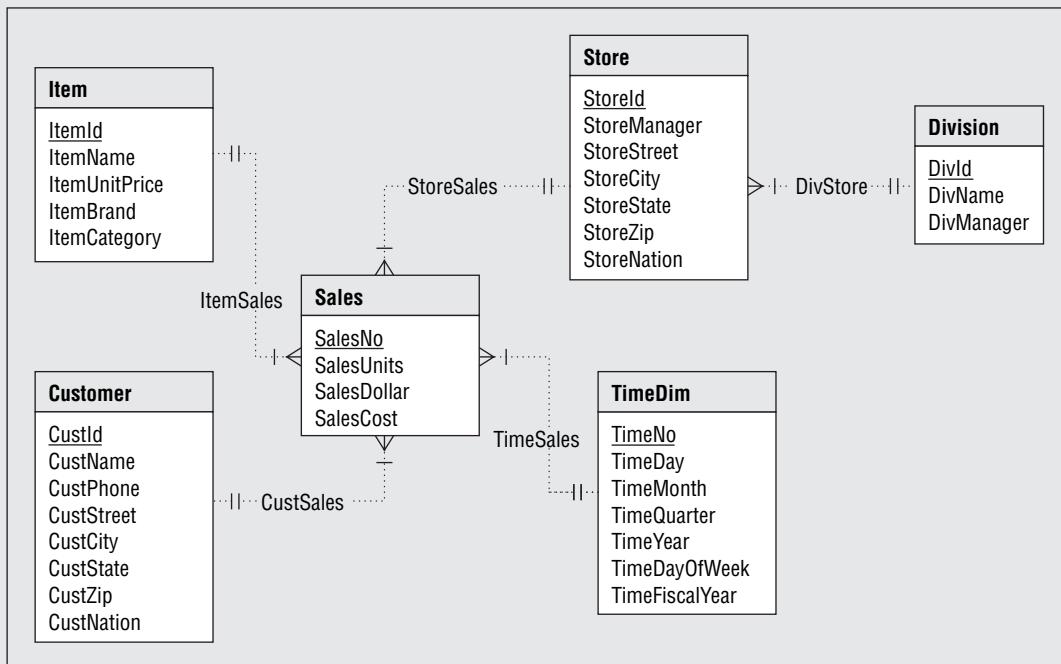
Llave primaria: ClaimNo  
 Llaves foráneas: IAVIN, ClmtNo, TPSSN

1. Identifique las dimensiones y medidas de un cubo de datos para el análisis de las pólizas de automóviles.
2. Trace un esquema de estrella o de copo de nieve para soportar las dimensiones y medidas en el cubo de datos del problema 1. Tal vez quiera utilizar la desnormalización para reducir el número de tablas de dimensión y las relaciones entre las tablas de dimensión.
3. Identifique las dimensiones y medidas en un cubo de datos para el análisis de las quejas.
4. Trace un esquema de estrella o de copo de nieve para soportar las dimensiones y medidas en el cubo de datos del problema 3. Tal vez quiera usar la desnormalización para reducir el número de tablas de dimensión y las relaciones entre las tablas de dimensión.
5. En el esquema de las pólizas, ¿qué nivel de detalle debe almacenarse? ¿Cuál debe ser el nivel más fino en un cubo de datos?
6. Identifique las jerarquías en las dimensiones del cubo de datos para la transacción de pólizas.
7. En el esquema de quejas, ¿qué nivel de detalle debe almacenarse? ¿Cuál debe ser el nivel más fino en un cubo de datos?
8. Identifique las jerarquías en las dimensiones del cubo de datos para el análisis de las quejas.
9. Describa el cubo de datos que resulta de la operación de dividir el cubo de datos de pólizas mediante un agente determinado.
10. Describa el cubo de datos que resulta de la operación de cortar el cubo de datos resultado de la operación de corte en el problema 9 por partes aseguradas con un código postal en un estado específico.
11. Empiece con un cubo de datos con cuatro dimensiones (*InsuredParty*, *InsuredAuto*, *Item* y *Agent*) y una medida (cantidad de pólizas) en las celdas. A partir de este cubo de datos, describa la operación para generar un nuevo cubo de datos con tres dimensiones (*InsuredParty*, *Item* y *Agent*) y una medida (cantidad promedio de pólizas de automóviles).
12. Identifique los niveles de dimensión y las jerarquías dentro de la tabla *Agent*. No necesita utilizar el enunciado CREATE DIMENSION de Oracle.
13. Identifique los niveles de dimensión y las jerarquías dentro de la tabla *InsuredParty*. No necesita utilizar el enunciado CREATE DIMENSION de Oracle.
14. Identifique los niveles de dimensión y las jerarquías dentro de la tabla *InsuredAuto*. No necesita utilizar el enunciado CREATE DIMENSION de Oracle.
15. ¿Es necesario tener una tabla de dimensión de tiempo para el data warehouse de la aseguradora de autos? Favor de justificar su respuesta y explicar cómo se representa el tiempo si no es en una tabla de dimensión independiente.
16. Para la tabla de dimensión *InsuredParty*, analice la estabilidad de las columnas en la tabla. ¿Qué columnas deben cambiar juntas? ¿De qué columnas deseariamos tener el historial?
17. Para la tabla de dimensión *InsuredAuto*, analice la estabilidad de las columnas en la tabla. ¿Qué columnas deben cambiar juntas? ¿De qué columnas deseariamos tener el historial?
18. Modifique la tabla de dimensión *InsuredParty* para un historial de la columna *IPRisk*. Proporcione una representación tipo II y otra representación tipo III con los valores de riesgo actuales y previos.
19. Modifique la tabla de dimensión *InsuredParty* para un historial limitado de las columnas *IPCity*, *IPState* e *IPZip*. El historial limitado debe registrar los valores actuales y previos, así como cambiar las fechas para la combinación de columnas.
20. Modifique la tabla de dimensión *InsuredParty* para un historial ilimitado de las columnas *IPCity*, *IPState* e *IPZip*. El historial ilimitado debe registrar las fechas de los cambios para la combinación de columnas, no para cada columna.

## Parte 2: Problemas de ventas en una tienda

La parte 2 le permite practicar con la manipulación de bases de datos relacionales de datos multidimensionales. Las preguntas en la parte 2 comprenden el esquema de copo de nieve de las ventas en una tienda utilizado en la sección 16.3. Como referencia, la figura 16.P1 muestra el ERD para el esquema de copo de nieve de las ventas de la tienda. Para soportar el uso de Oracle 10g con estos problemas, la sección del estudiante del sitio web de este libro contiene los enunciados CREATE TABLE de Oracle y datos de muestra para las tablas del esquema de ventas de la tienda.

1. Escriba un enunciado SELECT para resumir las ventas por estado en que está la tienda, año y marca de los artículos. El resultado debe calcular la SUMA de las ventas en dólares para los años 2005 y 2006. El resultado debe incluir totales completos para cada combinación de campos agrupados.

**FIGURA 16.P1** Esquema de copo de nieve ERD para el ejemplo de ventas de una tienda

2. Escriba un enunciado SELECT para resumir las ventas por año, trimestre y mes. El resultado debe calcular la SUMA de las ventas en dólares para los años 2005 y 2006. El resultado debe incluir totales parciales en el orden de los campos agrupados (año, trimestre y mes).
3. Escriba un enunciado SELECT para resumir las ventas por estado en que está la tienda, mes y año. El resultado debe calcular la SUMA de las ventas en dólares para los años 2005 y 2006. El resultado debe incluir totales parciales en el orden de la dimensión jerárquica (año y mes). No utilice el operador GROUPING SETS en su enunciado SQL.
4. Escriba un enunciado SELECT para resumir las ventas por el estado donde vive el cliente, código postal del cliente, año y trimestre. El resultado debe calcular la SUMA de las ventas en dólares para los años 2005 y 2006. El resultado debe incluir totales parciales para las dimensiones jerárquicas (año/trimestre y estado/código postal). No utilice el operador GROUPING SETS en su enunciado SQL.
5. Vuelva a escribir la solución del enunciado SQL para el problema 1 sin usar los operadores CUBE, ROLLUP ni GROUPING SETS. Para reducir el tiempo, puede escribir los primeros bloques de consultas y luego indicar el patrón para volver a escribir el resto de los bloques de consultas. Al volver a escribir las consultas, puede usar dos comillas (sin nada en el centro) como valor de texto predeterminado y 0 como el valor numérico predeterminado.
6. Vuelva a escribir la solución del enunciado SQL para el problema 2 sin usar los operadores CUBE, ROLLUP ni GROUPING SETS. Al volver a escribir las consultas, puede usar dos comillas (sin nada en el centro) como el valor de texto predeterminado y 0 como el valor numérico predeterminado.
7. Vuelva a escribir la solución del enunciado SQL para el problema 3 sin usar los operadores CUBE, ROLLUP ni GROUPING SETS. Al volver a escribir las consultas, puede usar dos comillas (sin nada en el centro) como el valor de texto predeterminado y 0 como el valor numérico predeterminado.
8. Vuelva a escribir la solución del enunciado SQL para el problema 3 utilizando el operador GROUPING SETS en lugar del operador ROLLUP.
9. Vuelva a escribir la solución del enunciado SQL para el problema 4 utilizando el operador GROUPING SETS en lugar del operador ROLLUP.
10. Realice el cálculo indicado y muestre la fórmula subyacente para los siguientes problemas. Entre paréntesis se muestra el número de valores únicos en cada dimensión.
  - Calcule el número máximo de filas para una consulta con un ascenso de año (2), trimestre (4) y mes (12). Separe el cálculo para mostrar el número de filas que aparecen en el resultado GROUP BY normal, así como el número de filas de subtotal generadas por el operador ROLLUP.

- Calcule el número máximo de filas para una consulta con un ascenso de año (2), trimestre (4), mes (12) y semanas por mes (4). Separe el cálculo para mostrar el número de filas que aparecen en el resultado GROUP BY normal, así como el número de filas de subtotal generadas por el operador ROLLUP.
  - Calcule el número máximo de filas para una consulta con un cubo de estado (5), marcas (10) y año (2). Separe el cálculo para mostrar el número de filas que aparecen en el resultado GROUP BY normal, así como el número de filas de subtotal generadas por el operador CUBE.
  - Calcule el número de grupos de subtotal en una consulta con un cubo de estado (5), división (4), marca (10) y año (2). Un grupo de subtotal equivale a un enunciado SELECT al formular la consulta sin ningún operador GROUP BY.
11. Escriba un enunciado CREATE DIMENSION de Oracle para una dimensión de clientes que consista en el identificador del cliente, la ciudad, el estado, el código postal y el país. Defina dos jerarquías que agrupen el identificador del cliente con la ciudad, el estado y el país, así como el identificador del cliente con el código postal, el estado y el país.
  12. Escribe un enunciado CREATE DIMENSION de Oracle para una dimensión de tiempo que consista en el identificador de tiempo, el día, el mes, el trimestre, el año, el año fiscal y el día de la semana. Defina tres jerarquías que agrupen el identificador de tiempo, el día, el mes, el trimestre y el año; así como el año, el identificador de tiempo y el año fiscal, además del identificador de tiempo y el día de la semana.
  13. Escriba un enunciado CREATE MATERIALIZED VIEW de Oracle para soportar el esquema de las ventas de una tienda. La vista materializada debe incluir la suma de las ventas en dólares y la suma del costo de las ventas. La vista materializada debe resumir las medidas por el identificador de la tienda, el identificador del artículo y el número del tiempo. La vista materializada debe incluir las ventas para el año 2005.
  14. Escriba un enunciado CREATE MATERIALIZED VIEW de Oracle para soportar el esquema de las ventas de una tienda. La vista materializada debe incluir la suma de las ventas en dólares y la suma del costo de las ventas. La vista materializada debe resumir las medidas por el identificador de la tienda, el identificador del artículo y el número del tiempo. La vista materializada debe incluir las ventas para el año 2006.
  15. Vuelva a escribir la solución del enunciado SQL para el problema 1 utilizando las vistas materializadas en los problemas 10 y 11. Debe ignorar el operador CUBE en la solución del problema 1. Su enunciado SELECT debe hacer referencia a las vistas materializadas, así como a las tablas de base, en caso de ser necesario.
  16. Vuelva a escribir la solución del enunciado SQL para el problema 1 utilizando las vistas materializadas en los problemas 10 y 11. Su enunciado SELECT debe hacer referencia a las vistas materializadas, así como a las tablas de base, en caso de ser necesario. Debe pensar con cuidado sobre la forma de manejar el operador CUBE en la consulta que vuela a escribir.
  17. Vuelva a escribir la solución del enunciado SQL para el problema 3 utilizando las vistas materializadas en los problemas 10 y 11. Debe ignorar el operador ROLLUP en la solución del problema 3. Su enunciado SELECT debe hacer referencia a las vistas materializadas, así como a las tablas de base, en caso de ser necesario.
  18. Vuelva a escribir la solución del enunciado SQL para el problema 3 utilizando las vistas materializadas en los problemas 10 y 11. Su enunciado SELECT debe hacer referencia a las vistas materializadas, así como a las tablas de base, en caso de ser necesario. Debe pensar con cuidado sobre la forma de manejar el operador ROLLUP en la consulta que vuelva a escribir.
  19. Escriba un enunciado CREATE MATERIALIZED VIEW de Oracle para soportar el esquema de ventas de una tienda. La vista materializada debe incluir la suma de las ventas en unidades y la suma del costo de las ventas. La vista materializada debe resumir las medidas por el código postal del cliente y el año de las ventas. La vista materializada debe incluir las ventas para el año 2005 y antes.
  20. Escriba un enunciado CREATE MATERIALIZED VIEW de Oracle para soportar el esquema de ventas de una tienda. La vista materializada debe incluir la suma de las ventas unitarias y la suma del costo de las ventas. La vista materializada debe resumir las medidas por el código postal del cliente, el año de las ventas y el trimestre de las ventas. La vista materializada debe incluir sólo las ventas en Estados Unidos.
  21. Escriba un enunciado CREATE MATERIALIZED VIEW de Oracle para soportar el esquema de las ventas de una tienda. La vista materializada debe incluir la suma de las ventas unitarias y la suma del costo de las ventas. La vista materializada debe resumir las medidas por el código postal del cliente, el

año de las ventas y el trimestre de las ventas. La vista materializada debe incluir las ventas en Canadá durante 2005 y 2006.

22. Escriba un enunciado SELECT utilizando las tablas base del data warehouse para recuperar la suma del costo de las ventas dividido entre la suma de las ventas unitarias en Estados Unidos y Canadá durante 2005. El resultado debe incluir el código postal del cliente, el año, y la suma del costo de las ventas por unidad. Vuelva a escribir el enunciado SELECT utilizando una o más de las vistas materializadas definidas en los problemas 19 a 21.
23. Escriba un enunciado SELECT utilizando las tablas base del data warehouse para recuperar la suma del costo de las ventas dividido entre la suma de las ventas unitarias en Estados Unidos y Canadá entre 2004 y 2006. El resultado debe incluir el código postal del cliente, el año y la suma del costo de las ventas por unidad. Vuelva a escribir el enunciado SELECT utilizando una o más de las vistas materializadas definidas en los problemas 19 a 21.

## Referencias para ampliar su estudio

Aunque este capítulo ofrece una introducción detallada a los data warehouse, tal vez quiera complementarlo con material especializado debido a la importancia y el alcance de la materia. Kimball (2002) e Inmon (2002), el padre de los data warehouse, han escrito libros muy leídos sobre el tema. Para profundizar en los detalles específicos de este capítulo deberá consultar Bouzeghoub *et al.* (1999) y Fisher y Berndt (2001) sobre el proceso de refrescamiento. Para detalles sobre las características del data warehouse de Oracle 10g, deberá consultar la documentación en línea en el sitio *Oracle Technology Network* (<http://www.oracle.com/technology>). Encontrará información adicional en sitios sobre temas como “Data warehouse” y “Minería de datos” en la sección de Recursos web del sitio web de este libro.



# Procesamiento cliente-servidor, procesamiento de bases de datos paralelas y bases de datos distribuidas

## Objetivos de aprendizaje

Este capítulo describe las formas en que los sistemas de gestión de bases de datos utilizan redes de computadoras y computadoras remotas para apoyar el procesamiento cliente-servidor, el procesamiento de bases de datos paralelas y las bases de datos distribuidas. Después de este capítulo, el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Listar razones para el procesamiento cliente-servidor, procesamiento de bases de datos paralelas y datos distribuidos.
- Describir arquitecturas de bases de datos cliente-servidor de dos niveles, tres niveles y niveles múltiples.
- Describir arquitecturas comunes para el procesamiento de bases de datos paralelas.
- Describir las diferencias entre la tecnología para bases de datos distribuidas firmemente integradas y con integración libre.
- Comparar los diferentes tipos de transparencia de bases de datos distribuidos.
- Comprender la naturaleza del procesamiento de consultas y el procesamiento de transacción para bases de datos distribuidas.

## Panorama general

---

Los capítulos 15 y 16 describieron el procesamiento de bases de datos para soporte de las transacciones y decisiones. Como se explicó en ambos capítulos, el procesamiento de apoyo a las transacciones y decisiones es vital para las organizaciones modernas. En este capítulo aprenderá cómo las redes de computadoras, las computadoras remotas y el almacenamiento de datos remoto pueden mejorar la confiabilidad y el rendimiento de ambos tipos de procesamiento.

Este capítulo explica las formas en que los DBMS utilizan redes de computadoras, computadoras remotas y almacenamiento de datos remoto. Antes de entender los detalles, debe entender la motivación para utilizar estos recursos. Este capítulo discute las razones empresariales para el procesamiento cliente-servidor, el procesamiento de bases de datos paralelas y los datos distribuidos. Después de comprender la motivación, está listo para aprender las arquitecturas para distribuir diferentes tareas en una organización cliente-servidor y dividir grandes cantidades de trabajo entre recursos con el uso de tecnología de bases de datos paralelas. La descripción del procesamiento de bases de datos paralelas en Oracle e IBM DB2 complementa la presentación conceptual. La distribución de los datos, además de la distribución del procesamiento, permite mayor flexibilidad, pero también implica más complejidad. Para bosquejar las negociaciones entre flexibilidad y complejidad, este capítulo explica las arquitecturas de bases de datos distribuidas, niveles de transparencia para datos distribuidos y procesamiento distribuido de bases de datos para consultas y transacciones. Los ejemplos de transparencia para bases de datos distribuidas de Oracle complementan el material conceptual.

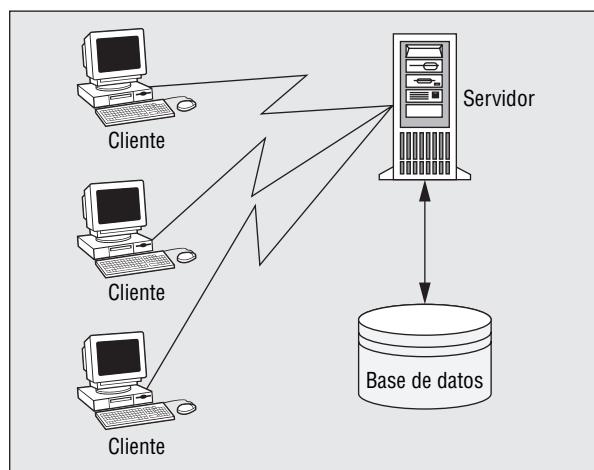
## 17.1 Panorama del procesamiento distribuido y de los datos distribuidos

Para entender este tema, es más fácil separar el procesamiento distribuido de los datos distribuidos. Ambas áreas tienen diferentes arquitecturas, problemas de diseño y tecnologías de procesamiento. Después de aprenderlos por separado, puede entender cómo combinarlos. Esta sección comienza su estudio discutiendo las motivaciones detrás de dos diferentes tipos de procesamiento distribuido (procesamiento cliente-servidor y de bases de datos paralelas) y bases de datos distribuidas.

### 17.1.1 Motivación para el procesamiento cliente-servidor

El enfoque cliente-servidor apoya el uso de recursos de cómputo remoto para realizar complejos procesos empresariales que consisten de una diversidad de subtareas. Por ejemplo, la compra electrónica es un proceso complejo que consiste en la selección del producto, levantamiento del pedido, gestión de inventarios, procesamiento de pago, embarque y regreso del producto. Un cliente es un programa que hace solicitudes a un servidor. El servidor ejecuta las solicitudes y comunica los resultados a los clientes. Los clientes y servidores pueden estar ordenados a través de computadoras en red para dividir el trabajo complejo en unidades más manejables. El ordenamiento más simple es dividir el trabajo entre clientes que procesen en computadoras personales y un servidor que procese en una computadora separada, como se muestra en la figura 17.1. La sección 17.2 presenta arquitecturas más poderosas para procesamiento cliente-servidor, junto con divisiones típicas del trabajo entre computadoras.

**FIGURA 17.1**  
Arquitectura cliente-servidor simple para procesamiento distribuido



El procesamiento distribuido con el enfoque cliente-servidor ofrece algunas ventajas relacionadas con la flexibilidad, escalabilidad e interoperabilidad. La flexibilidad se refiere a la facilidad de mantener y adaptar un sistema. Los costos de mantenimiento con frecuencia dominan el costo del desarrollo inicial de un sistema de información debido a la larga vida y revisiones del sistema. El enfoque cliente-servidor promueve la flexibilidad porque las secciones volátiles de código pueden aislarse de las secciones más estables. Por ejemplo, el código de interfaz de usuario puede separarse de las reglas del negocio y del código de acceso de datos. Si se despliega una nueva interfaz, otras partes del código permanecen sin cambio. Además, el enfoque cliente-servidor es idealmente adecuado para la programación orientada a objetos para apoyar el reuso. El capítulo 18 describe la programación y las bases de datos orientadas a objetos.

El enfoque cliente-servidor apoya el crecimiento escalable de la capacidad de hardware y software. La escalabilidad se refiere a la habilidad para agregar y remover capacidad en unidades pequeñas. La escalabilidad vertical se refiere a la habilidad para agregar capacidad en el lado del servidor. Por ejemplo, el trabajo proveniente de un servidor sobrecargado puede moverse hacia un nuevo servidor para aliviar un cuello de botella o manejar nueva demanda de estaciones de trabajo adicionales. El nuevo servidor puede tener justo el nivel de capacidad de computación adicional necesaria. La escalabilidad horizontal se refiere a la habilidad para agregar capacidad en el lado del cliente a través de estaciones de trabajo adicionales y movimiento de trabajo entre clientes y servidores. Por ejemplo, el trabajo puede moverse desde los clientes hacia un servidor para permitir el uso del hardware barato del cliente (clientes finos). El trabajo también puede moverse en la dirección opuesta (del servidor al cliente) para aliviar las cargas del procesamiento de servidor y sacar ventaja de las capacidades computacionales del cliente.

El crecimiento escalable también puede conducir al mejoramiento del desempeño. Por ejemplo, agregar middleware puede reducir los problemas de contención causados por muchos usuarios que acceden a una base de datos. La siguiente sección describe el middleware que puede gestionar eficientemente muchos usuarios simultáneos que acceden a una base de datos. Además, pueden emplearse servidores especializados para manejar el trabajo que de otro modo frenaría a todos los usuarios. Por ejemplo, los servidores multimedia pueden manejar solicitudes de imágenes, lo que libera a otros servidores de esta tarea consumidora de tiempo.

Los sistemas cliente-servidor basados en estándares abiertos apoyan la interoperabilidad. Interoperabilidad se refiere a la habilidad de dos o más sistemas para intercambiar y usar software y datos. Los estándares abiertos promueven un mercado de proveedores, lo que conduce a costos más bajos y mayor calidad. Los componentes de software en el mercado son interoperables si se conforman con los estándares. El área con mayor estandarización es Internet, donde las bases de datos cliente-servidor se vuelven cada vez más importantes.

A pesar de las ventajas del procesamiento cliente-servidor, pueden ocurrir algunos equívocos significativos. Desarrollar software cliente-servidor puede ser más complejo debido a las opciones arquitectónicas. Una arquitectura cliente-servidor especifica un ordenamiento de componentes y una división de procesamiento entre los componentes. La sección 17.2 presenta muchas posibles arquitecturas para procesamiento de bases de datos cliente-servidor. La elección de una arquitectura inadecuada puede conducir a un pobre rendimiento y a problemas de mantenimiento. Además de los conflictos arquitectónicos, el diseñador puede enfrentar una dificultad de decisión acerca de construir una base de datos cliente-servidor sobre métodos propietarios frente a estándares abiertos. Los métodos propietarios permiten una resolución de problemas más sencilla porque un proveedor es responsable de todos los problemas. Los métodos propietarios también pueden tener mejor rendimiento porque no son tan generales como los estándares abiertos. Sin embargo, a largo plazo, los métodos propietarios pueden ser costosos e inflexibles. Si un proveedor no crece con la industria, una base de datos cliente-servidor puede quedar obsoleta y su actualización puede resultar costosa.

### 17.1.2 Motivación para el procesamiento de bases de datos paralelas

En contraste con el uso del procesamiento cliente-servidor para distribuir el trabajo complejo entre computadoras en red, el procesamiento de bases de datos paralelas divide grandes tareas en

muchas tareas más pequeñas y las distribuye entre computadoras interconectadas. Por ejemplo, el procesamiento de bases de datos paralelas puede usarse para realizar una operación de unión en grandes tablas. El uso de las arquitecturas RAID descritas en el capítulo 8 es una forma simple de procesamiento de bases de datos paralelas. La sección 17.3 presenta arquitecturas más poderosas para procesamiento de bases de datos paralelas.

El procesamiento de bases de datos paralelas puede mejorar el rendimiento mediante escalamiento y aceleración. El escalamiento involucra la cantidad de trabajo que puede lograrse mediante el aumento de la capacidad de cómputo. El escalamiento mide el aumento en tamaño de una labor que puede realizarse mientras se mantiene el tiempo constante. El escalamiento ideal es lineal, en el que el aumento en capacidad computacional en  $n$  veces permite la conclusión de  $n$  veces la cantidad de trabajo en el mismo tiempo. El escalamiento lineal no es posible en la mayoría de las situaciones debido a los gastos de coordinación. El escalamiento se mide como la razón de la cantidad de trabajo concluido con la mayor configuración a la cantidad de trabajo concluido con la configuración original. Por ejemplo, si en la configuración original pueden procesarse 100 transacciones por minuto y cuando se duplica la capacidad pueden procesarse 175 transacciones por minuto, el escalamiento es de 1.75.

La aceleración implica la disminución en tiempo para completar una tarea en lugar de la cantidad de trabajo realizado. Con capacidad de computación añadida, la aceleración mide la reducción en tiempo mientras se mantiene la constante de tareas. Por ejemplo, las organizaciones usualmente requieren completar procesamientos diarios de regeneración de forma temporal para asegurar la disponibilidad para el siguiente día laboral. Las organizaciones necesitan determinar la cantidad de capacidad computacional adicional que garantizará la conclusión del trabajo dentro del tiempo permisible. La aceleración se mide mediante la razón del tiempo de conclusión con la configuración original al tiempo de conclusión con la capacidad adicional. Por ejemplo, si al duplicar la capacidad se reduce el tiempo de procesamiento de regeneración de 6 a 4 horas, la aceleración es de 1.5.

Disponibilidad es la accesibilidad de un sistema, usualmente medida como el tiempo productivo del sistema. Para computación de alta disponibilidad o resistente al fallo, un sistema experimenta poco tiempo improductivo y se recupera rápidamente de los fallos. La computación tolerante al fallo lleva la resistencia al límite en tanto que el procesamiento debe continuarse sin interrupción. El costo del tiempo improductivo determina el grado de disponibilidad deseada. El costo del tiempo improductivo puede incluir pérdidas de salarios, mano de obra perdida y equipo inactivo. Para una organización grande, el costo del tiempo improductivo puede ser de cientos a miles de dólares por hora. El procesamiento de bases de datos paralelas puede aumentar la disponibilidad porque un DBMS puede ajustarse dinámicamente al nivel de recursos disponibles. Las fallas de una computadora individual no detendrán el procesamiento en otras computadoras disponibles.

El gran inconveniente del procesamiento de bases de datos paralelas es el costo. El procesamiento de bases de datos paralelas implica grandes costos en software de DBMS y software de coordinación especializado. Existen posibles problemas de interoperabilidad, pues se requiere coordinación entre el software DBMS, el sistema operativo y los sistemas de almacenamiento. Los proveedores de DBMS ofrecen poderosas herramientas para desplegar y administrar la enorme complejidad del procesamiento de bases de datos paralelas. Si no se predicen mejoras al rendimiento, será un inconveniente significativo. Las mejoras predecibles en rendimiento permiten a las organizaciones planear capacidad adicional y ajustar dinámicamente la capacidad de acuerdo con los volúmenes de procesamiento anticipados y las restricciones en el tiempo de respuesta.

### 17.1.3 Motivación para datos distribuidos

Los datos distribuidos ofrecen algunas ventajas en relación con control de datos, costos de comunicación y rendimiento. Distribuir una base de datos permite la ubicación de datos de modo que se ajuste a la estructura de una organización. Por ejemplo, partes de una tabla de clientes pueden cargarse cerca de los centros de procesamiento de clientes. Las decisiones acerca de compartir y mantener los datos pueden establecerse localmente para proporcionar un control más cercano al uso de datos. Con frecuencia, los empleados y gerentes locales entienden mejor los conflictos con los datos que la gerencia en ubicaciones remotas.

**TABLA 17.1**  
Resumen de procesamiento distribuido y datos

Tecnología	Ventajas	Desventajas
Procesamiento cliente-servidor	Flexibilidad, interoperabilidad, escalabilidad	Enorme complejidad, alto costo de desarrollo, posibles problemas de interoperabilidad
Procesamiento de bases de datos paralelas	Aceleración, escalamiento, disponibilidad, escalabilidad para mejoras de rendimiento predictivas	Posibles problemas de interoperabilidad, alto costo
Bases de datos distribuidas	Control local de datos, rendimiento mejorado, costos de comunicación reducidos, confiabilidad aumentada	Enorme complejidad, preocupaciones adicionales por seguridad

Los datos distribuidos pueden conducir a menores costos de comunicación y rendimiento mejorado. Los datos deben ubicarse de modo que 80 por ciento de las solicitudes sean locales. Las solicitudes locales incurren en poco o ningún costo de comunicación y retardos, en comparación con las solicitudes remotas. La creciente disponibilidad de datos también puede conducir a rendimiento mejorado. Los datos son más accesibles porque no hay una sola computadora responsable de controlar el acceso. Además, los datos pueden copiarse de modo que estén disponibles en más de un sitio.

A pesar de las ventajas de los datos distribuidos, pueden ocurrir algunos problemas significativos. Los conflictos de diseño de bases de datos distribuidas son muy difíciles. Un pobre diseño puede conducir a enormes costos de comunicación y a pobre rendimiento. El diseño de bases de datos distribuidas es difícil debido a la carencia de herramientas, el número de opciones y las relaciones entre opciones. El procesamiento de transacción distribuida puede agregar considerables gastos, en especial para datos copiados. Los datos distribuidos implican más preocupaciones de seguridad porque son muchos los sitios que pueden manejar datos. Cada sitio debe protegerse adecuadamente de accesos no autorizados.

### 17.1.4 Resumen de ventajas y desventajas

Antes de seguir adelante, debe tomarse un momento para comparar el procesamiento distribuido y los datos distribuidos. La tabla 17.1 resume las ventajas y desventajas del procesamiento cliente-servidor, del procesamiento de bases de datos paralelas y de los datos distribuidos como tecnologías separadas. Para obtener un máximo aprovechamiento, las tecnologías pueden combinarse. En este momento, el procesamiento distribuido con el enfoque cliente-servidor es la tecnología más madura y ampliamente desplegada, aunque el procesamiento de bases de datos paralelas gana aceptación rápidamente. Conforme la tecnología de bases de datos distribuidas madure y gane aceptación, las organizaciones mostrarán las tres tecnologías.

## 17.2 Arquitecturas de bases de datos cliente-servidor

**arquitectura cliente-servidor**  
ordenamiento de componentes (clientes y servidores) entre computadoras conectadas mediante una red. Una arquitectura cliente-servidor soporta un eficiente procesamiento de mensajes (solicitudes de servicio) entre clientes y servidores.

El diseño de una base de datos cliente-servidor afecta las ventajas y desventajas citadas en la sección anterior. Un buen diseño tiende a amplificar las ventajas y a reducir las desventajas relacionadas con los requerimientos de una organización. Un pobre diseño puede exacerbar las desventajas y disminuir las ventajas. El diseño adecuado de una base de datos cliente-servidor puede hacer la diferencia entre el éxito y el fracaso de un proyecto de sistemas de información. Para ayudarlo a lograr buenos diseños, esta sección discute los conflictos de las bases de datos cliente-servidor y describe cómo pueden abordarse estos conflictos en varias arquitecturas.

### 17.2.1 Conflictos de diseño

Dos conflictos de diseño afectan el diseño de una base de datos cliente-servidor: división de procesamiento y gestión del proceso. La *división de procesamiento* se refiere a la asignación de tareas a clientes y servidores. La gestión del proceso implica interoperabilidad entre clientes y servidores y procesamiento eficiente de mensajes entre ellos. El software para gestión de proceso

se conoce como “middleware”, debido a su papel mediador. Esta sección describe estos conflictos, de modo que en la siguiente sección entenderá cómo los abordan diversas arquitecturas.

### *División de procesamiento*

En una base de datos cliente-servidor típica, hay algunas tareas que pueden realizarse localmente en un cliente o remotamente en un servidor. La siguiente lista describe brevemente estas tareas.

- **Presentación:** Código para mantener la interfaz de usuario gráfica. El código de presentación despliega objetos, monitoriza y responde a eventos. Los eventos incluyen acciones iniciadas por el usuario con el ratón y el teclado, así como eventos externos iniciados por cronómetros y otros usuarios.
- **Validación:** Código para asegurar la consistencia de la base de datos y entradas del usuario. La lógica de validación usualmente se expresa como reglas de integridad que se almacenan en una base de datos.
- **Lógica empresarial:** Código para realizar funciones empresariales, tales como cálculos de pagos, requisitos de elegibilidad y cálculo de intereses. La lógica empresarial puede cambiar conforme cambian los entornos regulador y competitivo.
- **Flujo de trabajo:** Código para asegurar la conclusión de los procesos empresariales. El código de flujo de trabajo puede enrutar formularios, enviar mensajes acerca de un plazo y notificar a los usuarios cuando un proceso empresarial se ha concluido.
- **Acceso a datos:** Código para extraer datos que respondan a consultas y modificaciones a la base de datos. El código de acceso a datos consiste en enunciados SQL y código de traducción, que usualmente es parte del DBMS. Si están involucradas múltiples bases de datos, algún código de traducción puede residir en software separado de un DBMS.

Parte de estas tareas pueden dividirse entre clientes y servidores. Por ejemplo, alguna validación puede realizarse en una PC cliente y otra puede realizarse en un servidor de base de datos. Por lo tanto, hay mucha flexibilidad acerca de cómo se pueden dividir las tareas de procesamiento. La sección 17.2.2 describe algunos modos populares para dividir las tareas de procesamiento.

### *Middleware*

La interoperabilidad es una función importante del *middleware*. Los clientes y servidores pueden existir en plataformas con diferente hardware, sistemas operativos, DBMS y lenguajes de programación. La figura 17.2 muestra el middleware que permite a los clientes y servidores comunicarse sin preocuparse por las plataformas subyacentes de los clientes y servidores. El middleware permite a un cliente y un servidor comunicarse sin conocimiento de la plataforma del otro.

Otra importante función del middleware es el control eficiente de mensajes. En un entorno cliente-servidor típico, existen muchos clientes que se comunican con unos cuantos servidores. Un servidor puede sobrecargarse sólo con gestionar los mensajes que recibe en vez de completar

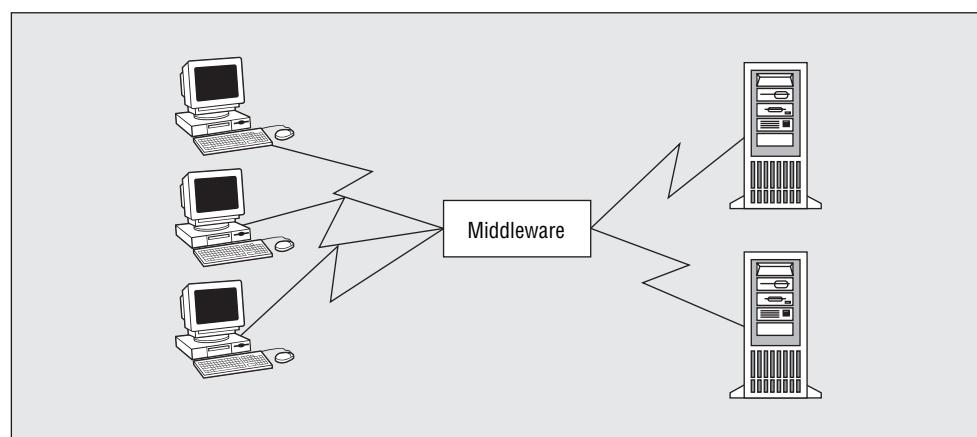
### **middleware**

componente de software en una arquitectura cliente-servidor que realiza gestión de procesos. El middleware permite a los servidores procesar eficientemente mensajes provenientes de una gran cantidad de clientes.

Además, el middleware puede permitir que los clientes y servidores se comuniquen a través de plataformas heterogéneas. El middleware para manejar grandes cargas de procesamiento con frecuencia se ubica en una computadora dedicada.

**FIGURA 17.2**

Computación cliente-servidor con middleware



las solicitudes. El middleware permite que los servidores se concentren en completar las solicitudes en vez de gestionarlas. El middleware puede realizar formación de colas, calendarización y enrutamiento de mensajes, al permitir a los clientes y servidores llevar a cabo trabajo a diferentes velocidades y tiempos.

Con base en las funciones de interoperabilidad y control de mensajes, están disponibles en el comercio muchos tipos de middleware, como se describe en la siguiente lista:

- **Los monitores de procesamiento de transacciones** son el tipo más antiguo de middleware. Tradicionalmente, los monitores de procesamiento de transacciones relevan al sistema operativo de la administración de los procesos de bases de datos. Un monitor de procesamiento de transacciones puede cambiar el control entre procesos mucho más rápido que un sistema operativo. En este papel, un monitor de procesamiento de transacciones recibe transacciones, las calendariza y gestiona hasta su conclusión. Recientemente, los monitores de procesamiento de transacciones han tomado tareas adicionales, como actualizar múltiples bases de datos en una sola transacción.
- **El middleware orientado a mensaje** mantiene una cola de mensajes. Un proceso cliente puede colocar un mensaje en una cola y un proceso servidor puede remover un mensaje de una cola. El middleware orientado a mensaje difiere de los monitores de procesamiento de transacción principalmente en la inteligencia de los mensajes. Los monitores de procesamiento de transacciones proporcionan inteligencia incorporada, pero usan mensajes simples. En contraste, el middleware orientado a mensaje proporciona mucho menos inteligencia incorporada, pero soporta mensajes más complejos.
- **Los agentes de solicitud de objetos** proporcionan un gran nivel de interoperabilidad e inteligencia de mensaje. Para usar un agente de solicitud de objetos, los mensajes deben codificarse en un lenguaje estándar de descripción de interfaz. Un agente de solicitud de objetos resuelve las diferencias de plataforma entre un cliente y un servidor. Además, un cliente puede comunicarse con un servidor sin saber la ubicación del servidor.
- **El middleware de acceso de datos** proporciona una interfaz uniforme a datos relacionales y no relacionales con el uso de SQL. Las solicitudes para acceso a datos desde un DBMS se envían a un controlador de acceso de datos en lugar de hacerlo directamente al DBMS. El controlador de acceso a datos convierte el enunciado SQL en el SQL soportado por el DBMS y luego enruta la solicitud al DBMS. El controlador de acceso a datos agrega otra capa de cabecera entre una aplicación y un DBMS. Sin embargo, el controlador de acceso a datos soporta independencia entre una aplicación y el SQL propietario soportado por un proveedor DBMS. Los dos middleware líderes para acceso a datos son el Open Database Connectivity (ODBC), soportado por Microsoft, y el Java Database Connectivity (JDBC), soportado por Oracle.

### 17.2.2 Descripción de arquitecturas

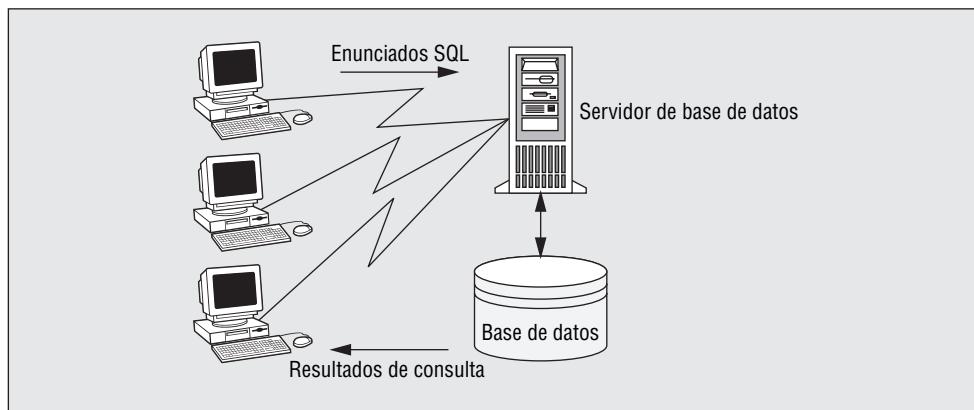
Los conflictos de diseño se abordan en muchas arquitecturas. Esta sección describe la división típica de procesamiento para cada arquitectura, enfoques de administración de mensajes y negociaciones entre arquitecturas.

#### *Arquitectura en dos niveles*

**arquitectura en dos niveles**  
arquitectura cliente-servidor en la que una PC cliente y un servidor de base de datos interactúan directamente para solicitar y transferir datos. La PC cliente contiene el código de interfaz de usuario, el servidor contiene la lógica de acceso a datos y la PC cliente y el servidor comparten las lógicas de validación y empresarial.

La arquitectura en dos niveles caracteriza a una PC cliente y un servidor de base de datos, como se muestra en la figura 17.3. La PC cliente contiene el código de presentación y los enunciados SQL para acceso a datos. El servidor de base de datos procesa los enunciados SQL y envía resultados de la consulta de vuelta a la PC cliente. Además, el servidor de base de datos realiza funciones de administración de procesos. Los códigos lógicos de validación y empresarial pueden dividirse entre la PC cliente y el servidor de base de datos. La PC cliente puede invocar procedimientos almacenados en el servidor de base de datos para validación y lógica empresarial. Por lo general, gran parte del código de lógica empresarial reside en el cliente. Las PC clientes en una arquitectura de dos niveles a veces se llaman “clientes gordos”, debido a la gran cantidad de lógica empresarial que contienen.

**FIGURA 17.3**  
**Arquitectura en dos niveles**



La arquitectura en dos niveles es un buen enfoque para sistemas con requerimientos establecidos y un número moderado de clientes. Por el lado positivo, la arquitectura en dos niveles es la más simple de implementar debido al número de buenos entornos de desarrollo comercial. Por el lado negativo, el mantenimiento de software puede ser difícil porque las PC clientes contienen una mezcla de códigos de lógica de presentación, validación y empresarial. Para hacer un cambio significativo en la lógica empresarial, el código debe modificarse en muchas PC clientes. Además, utilizar nueva tecnología puede ser difícil debido a que las arquitecturas en dos niveles con frecuencia se apoyan en software propietario en vez de en estándares abiertos. Para reducir la confianza en un servidor de base de datos particular, la PC cliente puede conectarse a controladores de bases de datos intermedios, como los controladores de Open Database Connectivity (ODBC) en lugar de directamente a un servidor de base de datos. Los controladores intermedios de bases de datos se comunican después con el servidor de base de datos.

El rendimiento puede ser pobre cuando una gran cantidad de clientes envían solicitudes, porque el servidor de base de datos puede estar abrumado con la gestión de los mensajes. Varias fuentes reportan que las arquitecturas en dos niveles están limitadas a aproximadamente 100 clientes simultáneos. Con un número más grande de clientes simultáneos puede ser necesaria una arquitectura en tres niveles. Además, conectarse a controladores intermedios en lugar de directamente a un servidor de base de datos puede disminuir el rendimiento.

#### *Arquitectura en tres niveles*

##### **arquitectura en tres niveles**

arquitectura cliente-servidor con tres capas: una PC cliente, un servidor de base de datos de parte interna y un middleware o servidor de aplicación.

##### **arquitectura en múltiples niveles**

arquitectura cliente-servidor con más de tres niveles: una PC cliente, un servidor de base de datos de parte interna, un servidor middleware interventor y servidores de aplicación. Los servidores de aplicación realizan lógica empresarial y gestionan tipos especializados de datos, como imágenes.

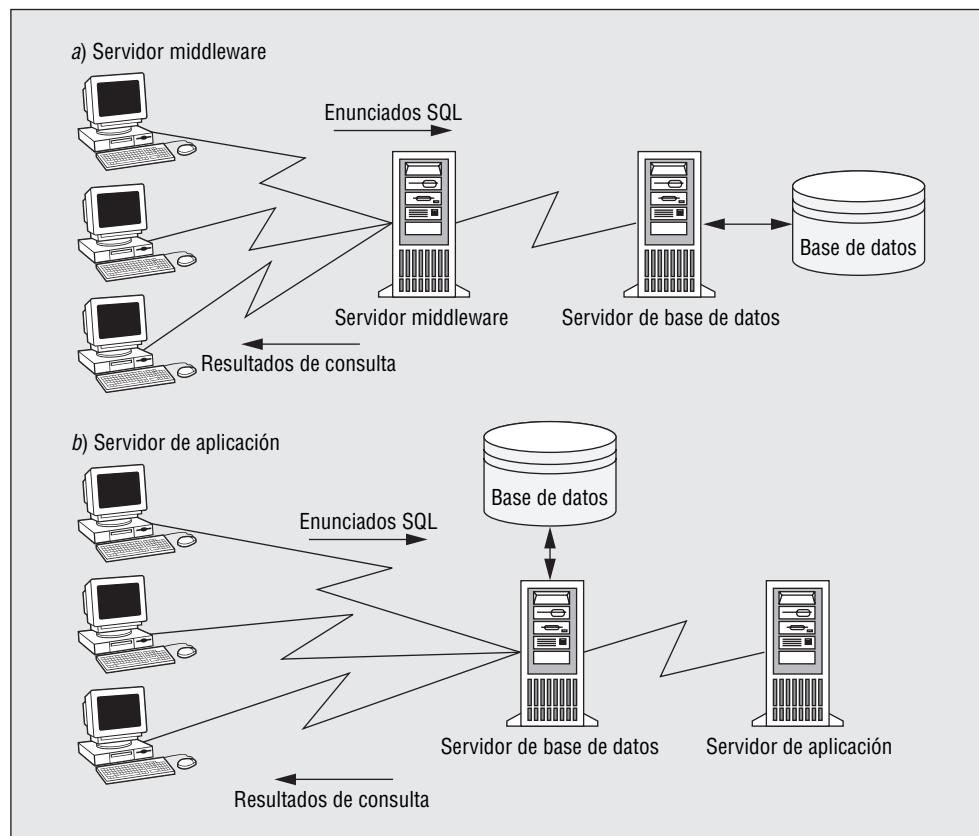
Para mejorar el rendimiento, la arquitectura en tres niveles agrega otra capa de servidor, como se muestra en la figura 17.4. Una forma de mejorar el rendimiento es agregar un servidor middleware [véase figura 17.4 a)] para manejar la administración del proceso. El middleware usualmente consiste en un monitor de procesamiento de transacción o middleware orientado a mensajes. Un monitor de procesamiento de transacción puede soportar más conexiones simultáneas que el middleware orientado a mensajes. Sin embargo, el middleware orientado a mensajes proporciona más flexibilidad en los tipos de mensajes soportados. Una segunda forma de mejorar el rendimiento es agregar un servidor de aplicación para tipos específicos de procesamiento, tales como escritura de reportes. En cualquier enfoque, el software adicional de servidor puede residir en una computadora separada, como se muestra en la figura 17.4. De manera alternativa, el software de servidor adicional puede distribuirse entre el servidor de base de datos y las PC clientes.

Aunque la arquitectura en tres niveles aborda las limitaciones de rendimiento de la arquitectura en dos niveles, no aborda las preocupaciones de la división del procesamiento. Las PC clientes y el servidor de base de datos todavía contienen la misma división de código, aunque las tareas del servidor de base de datos son reducidas. Las arquitecturas en múltiples niveles proporcionan más flexibilidad a la división de procesamiento.

#### *Arquitectura en múltiples niveles*

Para mejorar el rendimiento y proporcionar una división flexible de procesamiento, las arquitecturas en múltiples niveles soportan capas adicionales de servidores, como se muestra en la

**FIGURA 17.4**  
Arquitectura en tres niveles



**FIGURA 17.5**  
Arquitectura en tres niveles

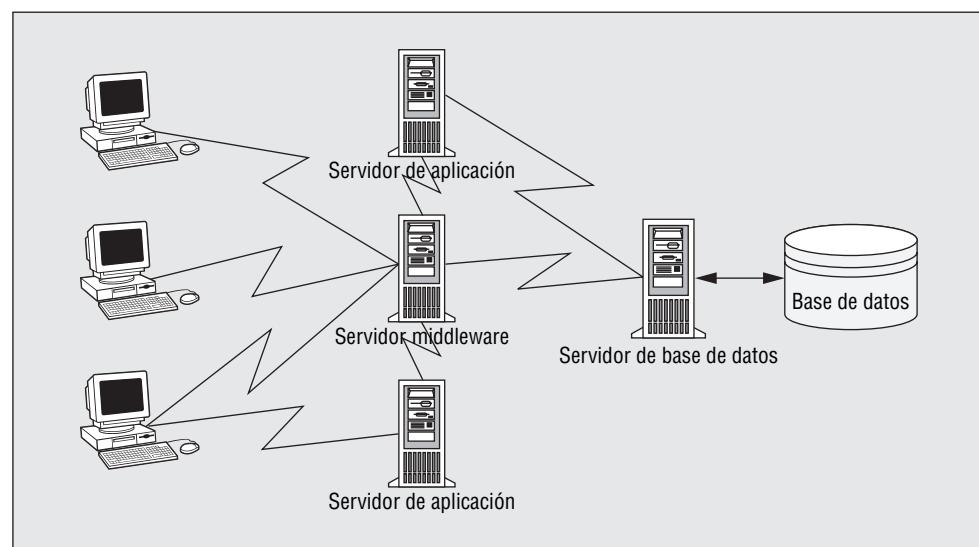
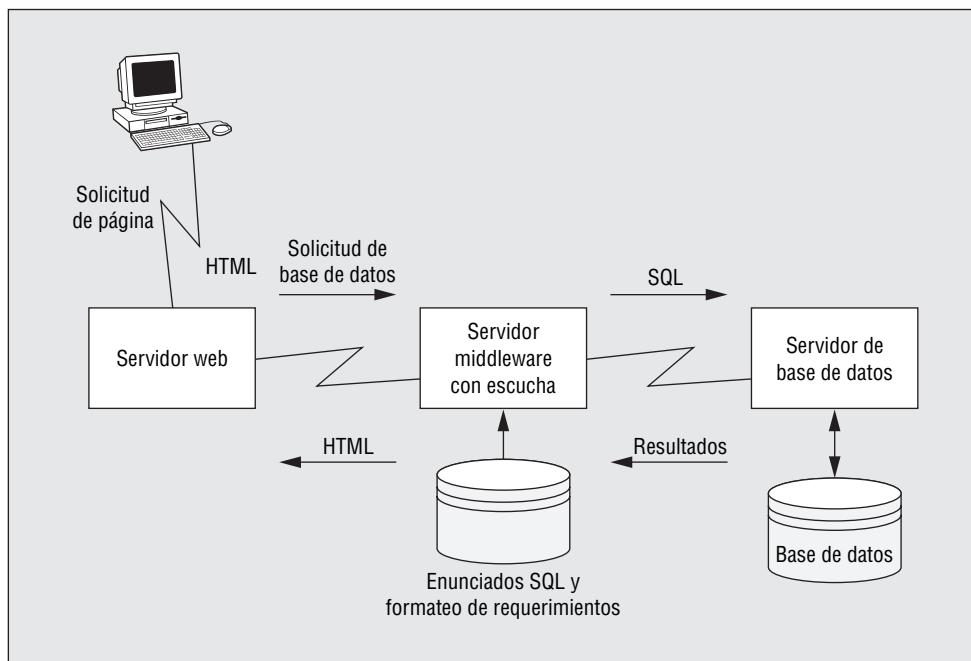


figura 17.5. Los servidores de aplicación pueden invocarse desde PC clientes, middleware y servidores de base de datos. Las capas del servidor adicional proporcionan una división más fina del procesamiento, que una arquitectura en dos o tres niveles. Además, las capas de servidor adicionales también pueden mejorar el rendimiento, porque pueden desplegarse middleware y servidores de aplicación.

Las arquitecturas en múltiples niveles para comercio electrónico implican un servidor web para procesar solicitudes de los navegadores web. El navegador y el servidor funcionan en conjunto para enviar y recibir páginas web escritas en Hypertext Markup Language (HTML).

**FIGURA 17.6**  
Servidor web que interactúa con el servidor middleware y el servidor de base de datos



Un navegador despliega páginas al interpretar el código HTML en el archivo que envía un servidor. El servidor web puede interactuar con un servidor middleware y de base de datos, como se muestra en la figura 17.6. Las solicitudes para acceso a base de datos se envían a un servidor middleware y luego se enrutan a un servidor de base de datos. Pueden agregarse servidores de aplicación para proporcionar niveles adicionales de procesamiento cliente-servidor.

#### *Arquitectura de servicios web*

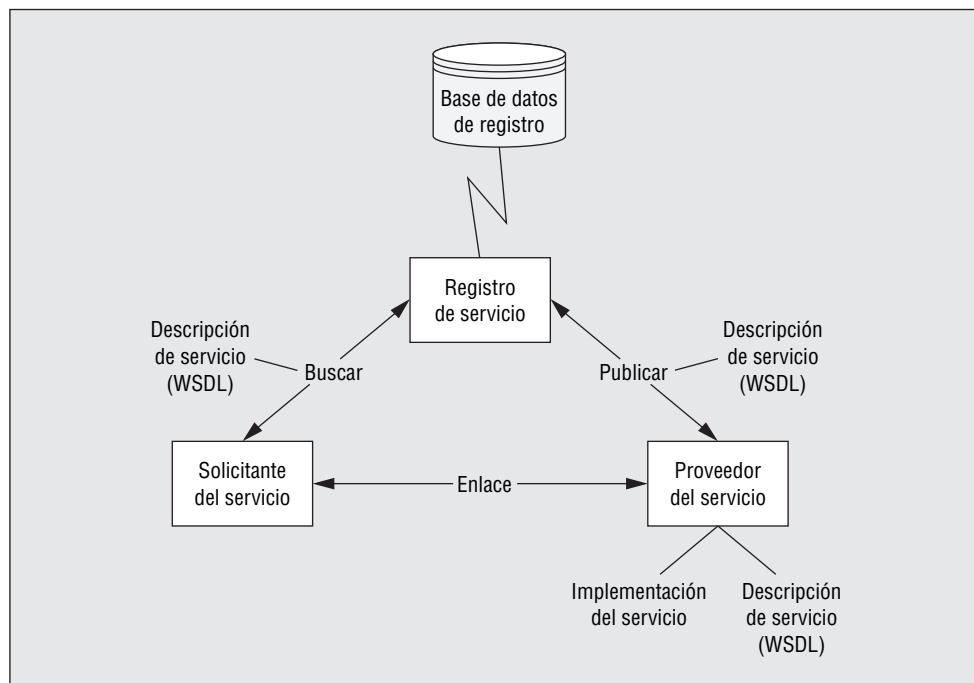
Los servicios web generalizan las arquitecturas en múltiples niveles para comercio electrónico con el uso de estándares de Internet para lograr alta interoperabilidad. El comercio electrónico implica servicios entre organizaciones proporcionados por agentes automatizados. Los servicios web permiten que las organizaciones reduzcan el costo del negocio electrónico al desplegar los servicios más rápido, comunicar nuevos servicios en formatos estándar y encontrar los servicios existentes de otras organizaciones. Los servicios web operan en Internet, una red de redes construida con lenguajes y protocolos estándar para alta interoperabilidad. Los servicios web usan estándares generales de Internet y nuevos estándares para comercio electrónico.

La arquitectura de servicios web soporta interacción entre un proveedor de servicio, un solicitante de servicio y registro de servicio, como se muestra en la figura 17.7. El proveedor de servicio posee el servicio y proporciona la plataforma de computación que ofrece el servicio. La aplicación solicitante del servicio busca un servicio y lo usa después de descubrirlo. El registro de servicio es el contenedor donde el proveedor de servicio publica su descripción de servicio y el solicitante del servicio busca los servicios disponibles. Después de que un solicitante de servicio encuentra un servicio, el solicitante del servicio usa la descripción del servicio para enlazarse con el proveedor del servicio e invocar la implementación del servicio mantenida por el proveedor.

La arquitectura de servicios web usa una colección de estándares de Internet interrelacionados con el fin de soportar la interoperabilidad, según se muestra en la tabla 17.2. XML (eXtensible Markup Language) es el fundamento subyacente de la mayoría de los estándares. XML es un metalenguaje que soporta la especificación de otros lenguajes. En la arquitectura de servicios web, los estándares WSFL, UDDI, WSDL y SOAP son lenguajes que acatan XML. Para un solicitante y proveedor de servicio, el WSDL es el estándar que se usa directamente para solicitar y enlazar un servicio web. Un documento WSDL proporciona una interfaz a un servicio que permite al proveedor del servicio ocultar los detalles del otorgamiento del servicio.

**Arquitectura de servicios web**  
arquitectura que soporta comercio electrónico entre organizaciones. Conjunto de estándares relacionados de Internet que soportan alta interoperabilidad entre solicitantes de servicio, proveedores de servicio y registros de servicio. El estándar más importante es el lenguaje de descripción de servicios web, que usan los solicitantes de servicio, los proveedores de servicio y los registros de servicio.

**FIGURA 17.7**  
Arquitectura de servicios web



**TABLA 17.2**  
Resumen de estándares de Internet para servicios web

Estándar	Uso
Lenguaje de Flujo de Servicios Web (WSFL)	Especificación de reglas de flujo de trabajo para servicios
Descripción Universal, Integración de Descubrimiento (UDDI)	Especificación de un directorio de servicios web incluidas terminología y restricciones de uso
Lenguaje de Descripción de Servicios Web (WSDL)	Especificación de servicios web
Protocolo Simple de Acceso a Objeto (SOAP)	Envío y recepción de mensajes XML
http, FTP, TCP-IP	Red y conexiones

Para satisfacer un servicio web, un proveedor de servicio puede utilizar procesamiento cliente-servidor, procesamiento de bases de datos paralelas y bases de datos distribuidas. Los detalles del procesamiento están ocultos para el solicitante del servicio. La arquitectura de servicios web proporciona otra capa de middleware en la cual publicar, encontrar y ejecutar servicios entre negocios electrónicos.

### 17.3 Procesamiento de bases de datos paralelas

En los últimos cinco años, la tecnología de bases de datos paralelas ha ganado aceptación comercial para grandes organizaciones. La mayoría de las empresas proveedoras de DBMS y algunos DBMSs de código abierto soportan tecnología de base de datos paralela para satisfacer la demanda del mercado. Las organizaciones utilizan estos productos para darse cuenta de los beneficios de la tecnología de base de datos paralela (escalamiento, aceleración y disponibilidad), mientras administra posibles problemas de interoperabilidad. Esta sección describe arquitecturas de bases de datos paralelas para proporcionar un marco de referencia para la comprensión de los ofrecimientos comerciales por parte de empresas proveedoras de DBMS.

### 17.3.1 Conflictos de arquitecturas y diseño

#### DBMS paralelo

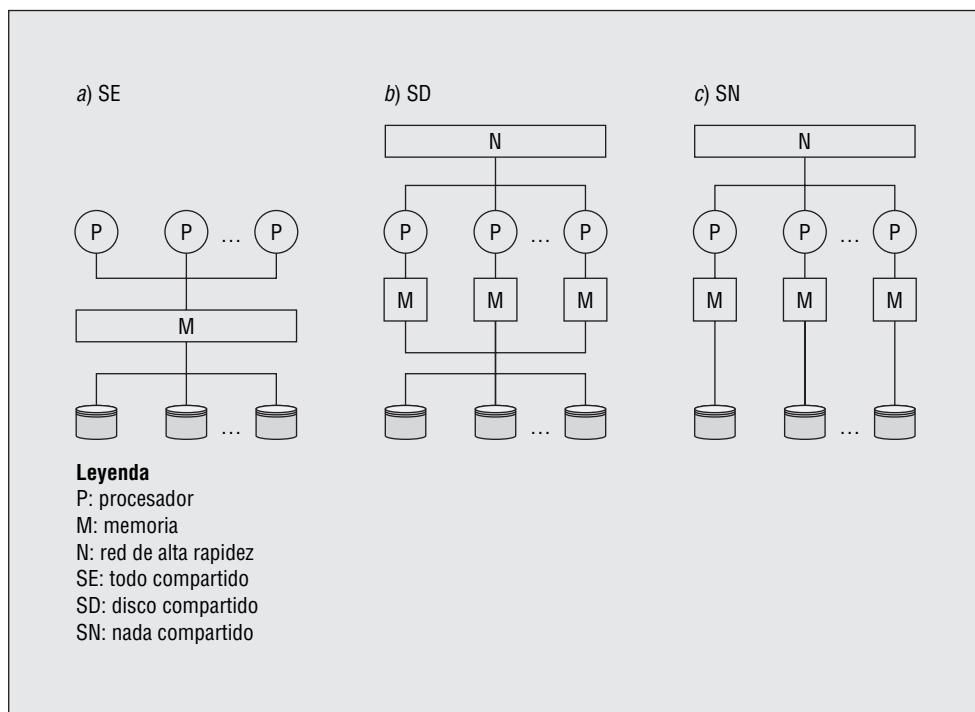
DBMS capaz de utilizar recursos de computación firmemente acoplados (procesadores, discos y memoria). El acoplamiento firme se logra mediante redes con tiempo de intercambio de datos comparable al tiempo de intercambio de datos con un disco. La tecnología de bases de datos paralelas promete mejoras en rendimiento y alta disponibilidad, aunque puede ocurrir problemas de interoperabilidad si no se gestiona adecuadamente.

Un DBMS paralelo usa una colección de recursos (procesadores, discos y memoria) para realizar trabajo en paralelo. Dado un presupuesto de recursos fijo, el trabajo se divide entre recursos para lograr niveles deseados de rendimiento (escalamiento y aceleración) y disponibilidad. Un DBMS paralelo usa los servicios de una red de alta rapidez, sistema operativo y sistema de almacenamiento para coordinar la división del trabajo entre recursos. Por ende, comprar un DBMS paralelo involucra una decisión acerca de todos estos componentes, no sólo del DBMS.

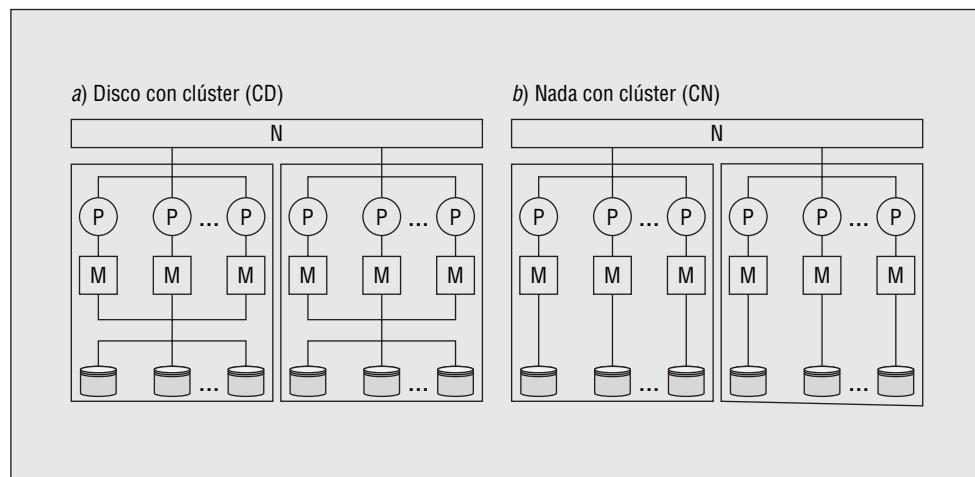
El grado de recursos compartidos determina las arquitecturas para procesamiento de bases de datos paralelas. El estándar de clasificación de arquitecturas se conoce como todo compartido (SE), discos compartidos (SD) y nada compartido (SN), como se muestra en la figura 17.8. En el enfoque SE, memoria y discos se comparten entre una colección de procesadores. El enfoque SE usualmente está relacionado como una sola computadora de multiprocesamiento y no como una arquitectura de base de datos paralela. En la arquitectura SD, cada procesador tiene su memoria privada, pero los discos se comparten entre todos los procesadores. En la arquitectura SN, cada procesador tiene su propia memoria y discos. En las arquitecturas SD y SE no es necesario el particionamiento, porque cada procesador tiene acceso a todos los datos.

Para flexibilidad adicional, las arquitecturas básicas se extienden mediante clustering. Un clúster es un acoplamiento firme de dos o más computadoras, de modo que se comparten como una sola computadora. La figura 17.9 extiende la figura 17.8 con clustering para mostrar las arquitecturas de disco con clúster (CD) y nada clúster (CN). En la arquitectura CD, los procesadores en cada clúster comparten todos los discos, pero nada se comparte a través de los clústeres. En la arquitectura CN, los procesadores en cada clúster no comparten recursos, pero se puede manipular cada clúster para trabajar en paralelo en la realización de una tarea. La figura 17.9 sólo muestra dos clústeres, pero el enfoque de clustering no se limita a ellos. Para mayor flexibilidad, pueden configurarse dinámicamente el número de clústeres y la membresía a ellos. Además, cada nodo de procesador en un clúster puede ser una computadora de multiprocesamiento o un procesador individual.

**FIGURA 17.8**  
Arquitecturas básicas de bases de datos paralelas



**FIGURA 17.9**  
Arquitecturas con clustering de bases de datos paralelas



### arquitecturas para procesamiento de bases de datos paralelas

Las arquitecturas disco con clúster (CD) y nada con clúster (CN) dominan en los DBMS comerciales. Un clúster es un acoplamiento firme de dos o más computadoras que se comportan como si fueran una sola. En la arquitectura CD, los procesadores en cada clúster comparten todos los discos, pero no se comparte nada a través de clústeres. En la arquitectura CN, los procesadores en cada clúster no comparten recursos, pero se puede manipular cada clúster para que trabaje en paralelo para realizar una tarea.

En todas las arquitecturas de bases de datos paralelas, el hecho de compartir recursos es transparente a las aplicaciones. El código de aplicación (SQL y enunciados en lenguaje de programación) no necesitan cambiarse para sacar ventaja del procesamiento de bases de datos paralelas. En contraste, las arquitecturas de bases de datos distribuidas que se presentan en la sección 17.4 usualmente no ofrecen un procesamiento transparente debido a las diferentes metas de las bases de datos distribuidas.

Los principales conflictos de diseño que influyen en el rendimiento de las arquitecturas de bases de datos paralelas son equilibrio de carga, coherencia de caché y comunicación interprocesadores. El equilibrio de carga involucra la cantidad de trabajo asignado a diferentes procesadores en un clúster. Lo ideal es que cada procesador tenga la misma cantidad de trabajo para utilizar el clúster por completo. La arquitectura CN es más sensible al equilibrio de carga debido a la necesidad de particionado de datos. Puede ser difícil dividir un subconjunto de una base de datos para lograr igual división de trabajo debido a que el sesgo de datos es común entre las columnas de bases de datos.

La coherencia de caché implica sincronización entre memorias locales y almacenamiento de disco común. Después de que un procesador direcciona una página de disco, la imagen de esta página permanece en la caché asociada con un procesador dado. Una inconsistencia ocurre si otro procesador cambia la página en su propio búfer. Para evitar inconsistencias, cuando se accede a una página de disco debe hacerse una verificación de otros cachés locales para coordinar los cambios producidos en los cachés de dichos procesadores. Por definición, el problema de coherencia de caché se limita a las arquitecturas de disco compartido (SD y CD).

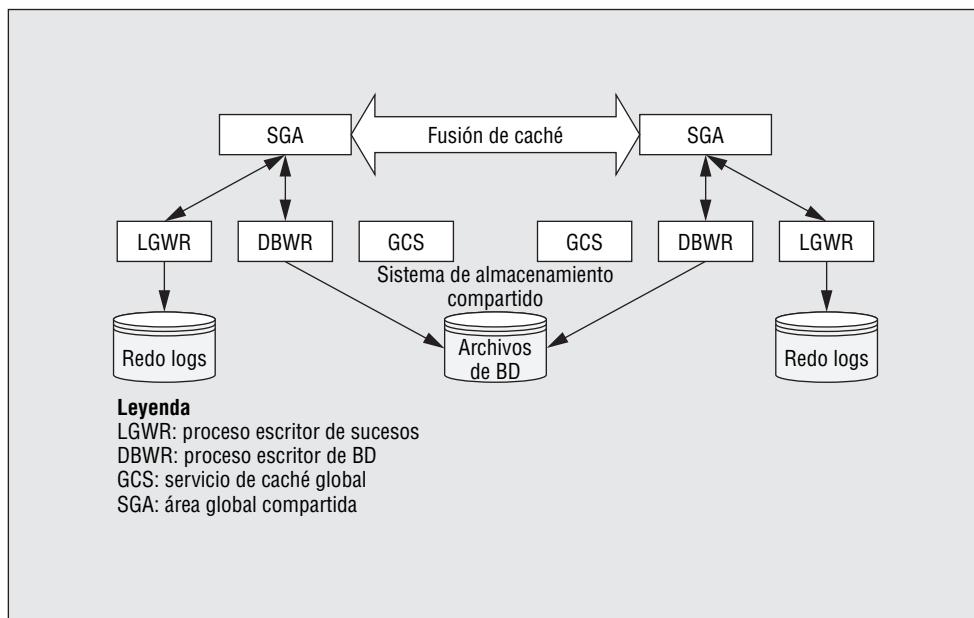
La comunicación interprocesadores involucra los mensajes generados para sincronizar acciones de procesadores independientes. El paralelismo particionado, como se usa para las arquitecturas nada compartido, puede crear grandes cantidades de comunicación interprocesadores. En particular, las operaciones conjuntas particionadas pueden generar una gran cantidad de comunicación elevada para combinar resultados parciales ejecutados en diferentes procesadores.

La investigación y el desarrollo de la tecnología de bases de datos paralelas encontraron soluciones razonables a los problemas de equilibrio de carga, coherencia de caché y comunicación interprocesadores. Los proveedores comerciales de DBMS han incorporado soluciones en su oferta de DBMS. Comercialmente, dominan las arquitecturas CD y CN. La siguiente subsección muestra DBMS comerciales que usan las arquitecturas CD y CN.

### 17.3.2 Tecnología comercial de bases de datos paralelas

Esta sección presenta detalles de DBMS comerciales que ofrecen DBMS paralelos. Debido a que la clasificación discutida en la sección 17.3.1 no es suficiente para entender las diferencias entre productos reales, esta sección proporciona detalles acerca de dos DBMS paralelos

**FIGURA 17.10**  
Ejemplo de clúster de dos nodos de Oracle



sobresalientes. Aunque ambos DBMS prometen grandes niveles de rendimiento, escalabilidad y disponibilidad, las negociaciones entre niveles son inevitables dados los enfoques contrastantes de los dos enfoques DBMS paralelos.

#### *Oracle Real Application Clusters*

Oracle Real Application Clusters (RAC) requiere un clúster de hardware subyacente, un grupo de servidores independientes que cooperan como un solo sistema. Los principales componentes de clúster son los nodos de procesador, un clúster de interconexión y un subsistema de almacenamiento compartido, como se muestra en la figura 17.10. Cada nodo de servidor tiene su propia instancia de base de datos con un proceso de escritura de registro, un proceso de escritura de base de datos y una área global compartida que contiene bloques de base de datos, búferes de reelaboración de registro, información de diccionario y enunciados SQL compartidos. Todos los procesos de escritura de bases de datos en un clúster usan el mismo sistema de almacenamiento compartido. Cada instancia de base de datos mantiene sus propios registros de recuperación.

La tecnología de fusión de caché en RAC permite el acceso sincronizado al caché a través de todos los nodos en un clúster, sin incurrir en costosas operaciones I/O de disco. El Servicio de Caché Global (GCS) es un componente RAC que implementa la tecnología fusión de caché. El GCS mantiene un directorio distribuido de recursos, como bloques de datos y colas de acceso a los recursos. Para optimizar el rendimiento se asigna un recurso al nodo con los accesos más frecuentes al recurso. El acceso a un recurso por otros nodos se controla mediante su nodo maestro.

Oracle RAC soporta las características que se citan a continuación. Todas estas características están disponibles sin base de datos o particionado de la aplicación.

- El optimizador de consulta considera el número de procesadores, grado de paralelismo de la consulta y carga de trabajo del CPU para distribuir la carga de trabajo entre los nodos en el clúster y para la utilización óptima de los recursos de hardware.
- El equilibrio de carga distribuye automáticamente las conexiones entre nodos activos en un clúster con base en la carga de trabajo del nodo. Las solicitudes de conexión se envían al nodo menos congestionado en un clúster.
- La sobrefalla automática permite una rápida conmutación de un nodo que falla a un nodo sobreviviente. La conmutación entre nodos puede planearse para permitir el mantenimiento periódico.

- El Oracle Enterprise Manager proporciona una interfaz de usuario comprensiva para gestión y configuración de clústeres. Despliega información de sesión de usuario para nodos en un clúster, rastreo de enunciado SQL y conexión de información entre usuario y máquina. Los administradores pueden establecer el nivel de prioridad de los diferentes usuarios y crear planes de recurso para grupos de usuarios.

#### *IBM DB2 Enterprise Server Edition con la opción DPF*

La Database Partitioning Feature (DPF) proporciona procesamiento paralelo estilo CN para bases de datos DB2. DB2 sin la opción DPF soporta procesamiento transparente de bases de datos paralelas para máquinas multiprocesador. Los planes de acceso DB2 pueden explotar todos los CPU y los discos físicos en un servidor multiprocesador. La opción DPF agrega la capacidad de particionar una base de datos a través de un grupo de máquinas. Una sola imagen de una base de datos puede abarcar múltiples máquinas y aún así aparecer a los usuarios y aplicaciones como una sola imagen de base de datos. El paralelismo particionado disponible con la opción DPF proporciona mucho mayor escalabilidad que el paralelismo multiprocesador solo.

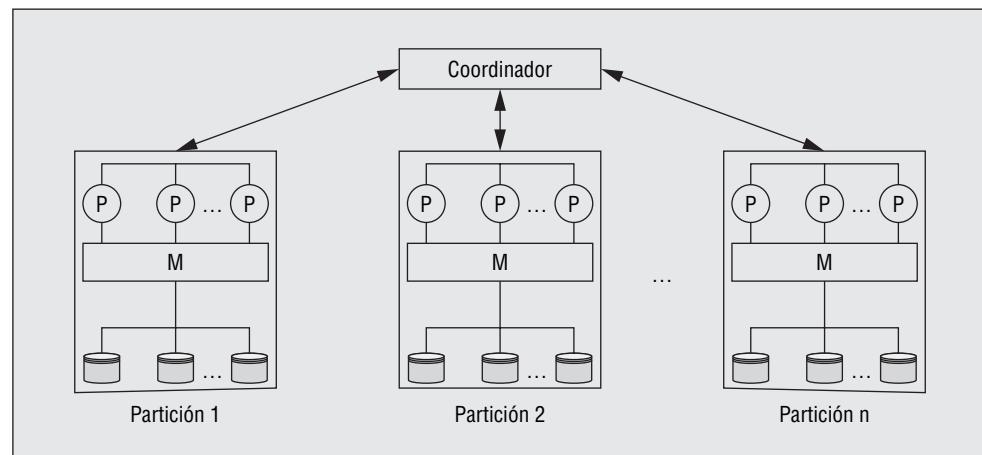
Particionar con la opción DPF es transparente a los usuarios y aplicaciones. La interacción con el usuario ocurre a través de una partición de base de datos, conocida como nodo coordinador para dicho usuario. La figura 17.11 muestra la coordinación en una base de datos particionada para nodos de servidor multiprocesador. La partición de base de datos a la que un cliente o aplicación se conecta se convierte en el nodo coordinador. Los usuarios deben distribuirse a través de servidores para distribuir la función de coordinador.

La opción DPF soporta particionado automático o DBA particionado determinado por DBA. Con el particionado automático, los datos se distribuyen entre particiones con el uso de un mapa de particiones actualizable y un algoritmo de hashing, que determina el lugar y recuperación de cada fila de datos. Para el particionado determinado por DBA, puede seleccionarse una columna como una llave de particionado. Con cualquier enfoque la colocación física de datos es transparente a los usuarios y aplicaciones.

La opción DPF de DB2 soporta las siguientes características. Todas éstas requieren particionado de bases de datos.

- El optimizador de consultas usa información acerca del particionado de datos para buscar planes de acceso. Mientras compara diferentes planes de acceso, el optimizador cuenta el paralelismo de diferentes operaciones y costos asociados con el envío de mensajes entre particiones de base de datos.
- DPF proporciona un mayor grado de escalabilidad a través de su soporte para miles de particiones. Para la opción DPF se han reportado mejoras de rendimiento casi lineales.
- El paralelismo particionado proporciona rendimiento de acceso mejorado porque cada partición mantiene sus propios archivos de registro.

**FIGURA 17.11**  
Coordinación para  
paralelismo particio-  
nado en DPF



## 17.4 Arquitecturas para sistemas de gestión de bases de datos distribuidas

---

Los DBMS distribuidos implican tecnología diferente a la del procesamiento cliente-servidor y al procesamiento de bases de datos paralelas. El procesamiento cliente-servidor enfatiza la distribución de funciones entre computadoras en red con el uso de middleware para gestión de proceso. Una diferencia fundamental entre el procesamiento de bases de datos paralelas y distribuidas es la autonomía. Las bases de datos distribuidas proporcionan autonomía de sitio mientras que las bases de datos paralelas, no. Por lo tanto, la base de datos distribuida requiere un conjunto diferente de características y tecnología.

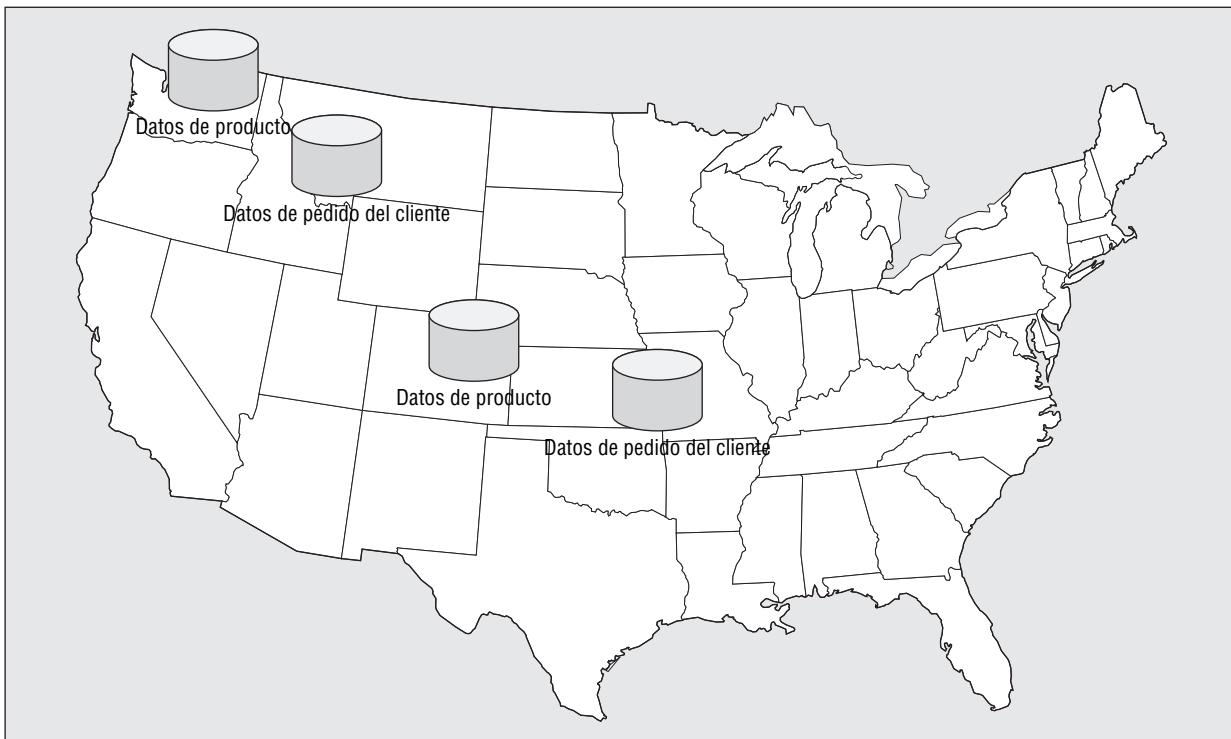
Para apoyar el procesamiento de bases de datos distribuidas es necesario extensiones fundamentales a un DBMS. Existen diferentes arquitecturas de componente subyacentes a las extensiones que gestionan solicitudes de bases de datos distribuidas y una arquitectura de esquema diferente que proporciona capas adicionales de descripción de datos. Esta sección describe la arquitectura de componente y la arquitectura de esquema para proporcionar el cimiento para más detalles acerca del procesamiento de bases de datos distribuidas en las secciones siguientes.

### 17.4.1 Arquitectura de componente

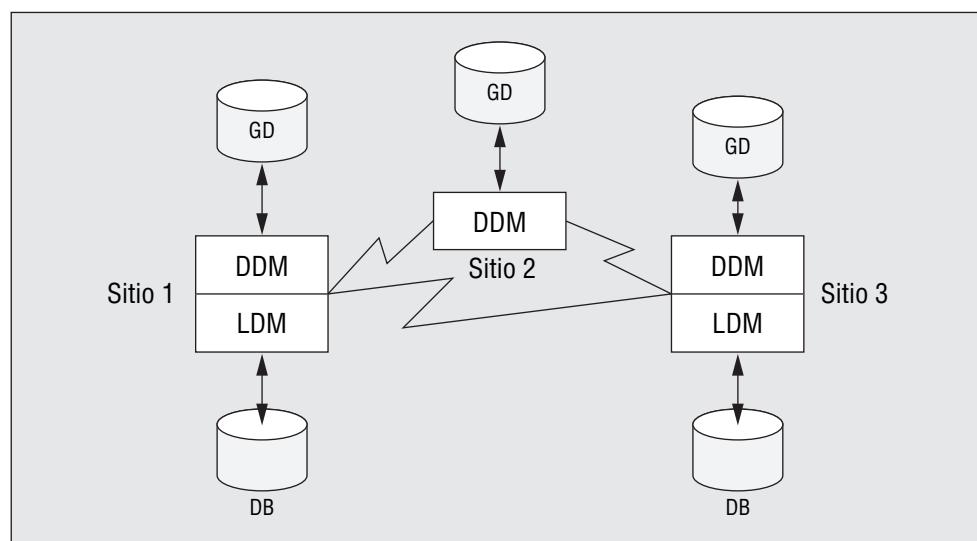
Los DBMS distribuidos soportan solicitudes globales que usan datos almacenados en más de un sitio autónomo. Un sitio es cualquier computadora controlada localmente con una dirección de red única. Los sitios con frecuencia están distribuidos geográficamente, aunque la definición soporta sitios ubicados en proximidad cercana. Las solicitudes globales son consultas que combinan datos desde más de un sitio y transacciones que actualizan datos en más de un sitio. Una solicitud global puede implicar una colección de enunciados que acceden a datos locales en algunos enunciados y a datos remotos en otros. Los datos locales están controlados por el sitio en el que un usuario normalmente se conecta. Los datos remotos implican un sitio diferente, en el que un usuario puede incluso no tener cuenta de acceso. Si todas las solicitudes requieren datos de un solo sitio, no se requiere capacidades de procesamiento de bases de datos distribuidas.

Para bosquejar las solicitudes globales, necesita comenzar con una base de datos distribuida. Las bases de datos distribuidas son potencialmente útiles para organizaciones que operan en múltiples ubicaciones con control local de recursos computacionales. La figura 17.12 muestra una base de datos distribuida para una compañía electrónica minorista. La compañía realiza procesamiento de clientes en Boise y Tulsa y administra bodegas en Seattle y Denver. La distribución de la base de datos sigue las ubicaciones geográficas del negocio. Las tablas *Customer*, *Order* y *OrderLine* (datos de pedido del cliente) están divididas entre Boise y Tulsa, mientras que las tablas *Product* e *Inventory* (datos de producto) están divididas entre Seattle y Denver. Un ejemplo de una consulta global es verificar ambos sitios de bodega para que la cantidad suficiente de un producto satisfaga un embarque. Un ejemplo de transacción global es un formato de entrada de pedido que inserte registros en las tablas *Order* y *OrderLine* en una ubicación, y actualice la tabla *Product* en el sitio de almacenamiento más cercano.

Para soportar las consultas y transacciones globales, los DBMS distribuidos contienen componentes adicionales en comparación con los DBMS tradicionales no distribuidos. La figura 17.13 muestra un posible ordenamiento de los componentes de un DBMS distribuido. Cada servidor con acceso a la base de datos distribuida se conoce como *sitio*. Si un sitio contiene una base de datos, ésta se controla con un administrador de datos local (LDM). Los administradores de datos locales proporcionan características completas de un DBMS, como se describe en otros capítulos. El administrador de datos distribuidos (DDM) optimiza la ejecución de consultas a través de sitios, coordina el control de concurrencia y la recuperación a través de sitios, y controla el acceso a datos remotos. Al realizar estas tareas, el DDM utiliza el diccionario global (GD) para localizar partes de la base de datos. El GD puede distribuirse en varios sitios de manera similar a como se distribuyen los datos. Debido a la complejidad del administrador de base de datos distribuida, la sección 17.6 presenta más detalles acerca del procesamiento de consulta distribuida y del procesamiento de transacciones.

**FIGURA 17.12** Distribución de datos de la solicitud de pedido**FIGURA 17.13**

Arquitectura de componente de un DBMS distribuido



En la arquitectura de componente, los administradores de base de datos locales pueden ser homogéneos o heterogéneos. Un DBMS distribuido con DBMS local homogéneo está *firamente integrado*.

El administrador de base de datos distribuido puede solicitar componentes internos y acceso al estado interno de los administradores de datos locales. La firme integración permite que el DBMS distribuido soporte eficientemente las consultas y transacciones distribuidas. Sin embargo, el requisito de homogeneidad prohíbe la integración de bases de datos existentes.

Un DBMS distribuido con administradores de datos locales heterogéneos tiene *integración floja*. El administrador de base de datos distribuido actúa como middleware para coordinar

administradores de datos locales. Con frecuencia, SQL proporciona la interfaz entre el administrador de datos distribuido y los administradores de datos locales. La integración floja soporta compartir datos entre sistemas heredados y organizaciones independientes. Sin embargo, el enfoque de integración floja puede no ser capaz de soportar procesamiento de transacciones de forma confiable y eficiente.

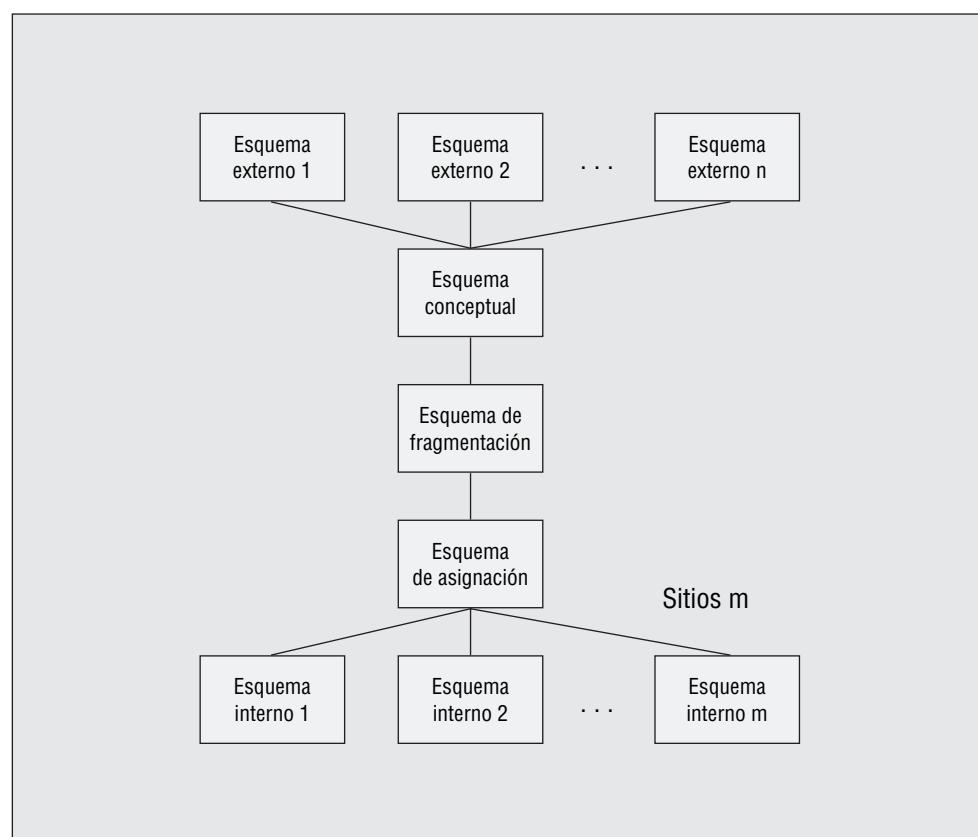
### 17.4.2 Arquitecturas de esquema

Para acomodar la distribución de datos son necesarias capas adicionales de descripción de datos. Sin embargo, no hay una arquitectura de esquema ampliamente aceptada para bases de datos distribuidas, como lo es la ampliamente aceptada arquitectura de tres esquemas para los DBMS tradicionales. Esta sección presenta posibles arquitecturas de esquemas para DBMS distribuidos firmemente integrados y DBMS distribuidos con integración floja. Las arquitecturas proporcionan una referencia acerca de los tipos de descripción de datos necesarios y la forma de compartir la descripción de datos.

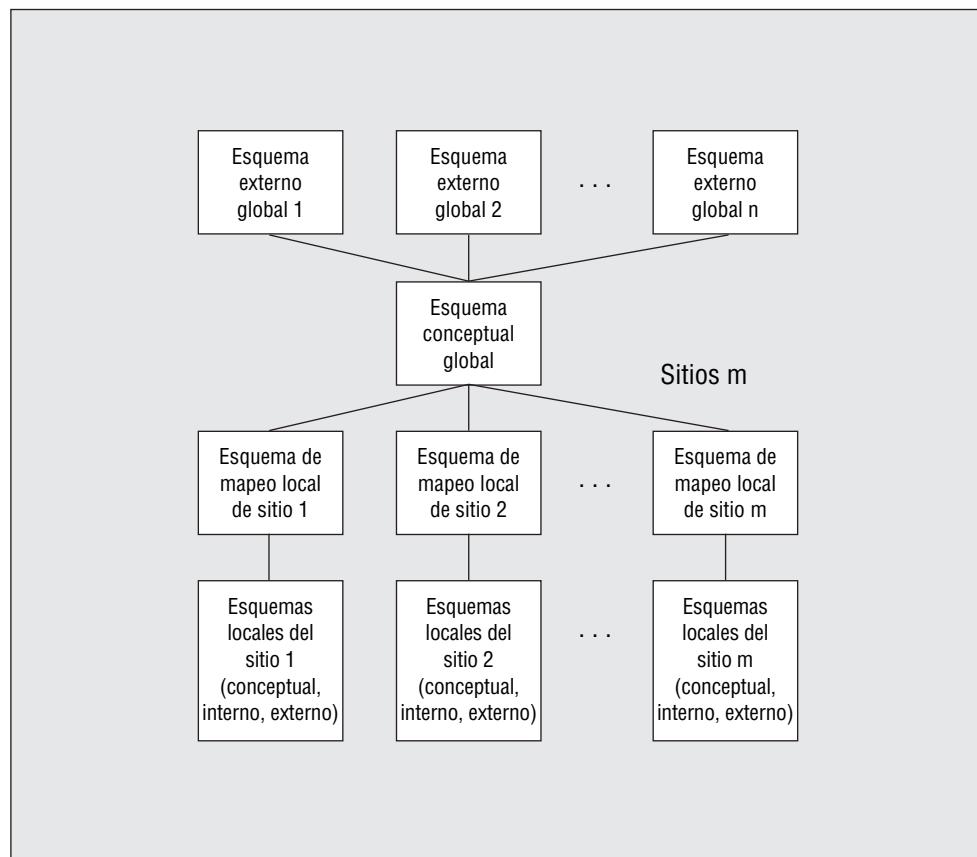
La arquitectura de esquema para un DBMS distribuido firmemente integrado contiene capas adicionales para fragmentación y asignación, como se muestra en la figura 17.14. El esquema de fragmentación contiene la definición de cada fragmento, mientras que el esquema de asignación contiene la ubicación de cada fragmento. Un fragmento puede definirse como un subconjunto vertical (operación de proyecto), un subconjunto horizontal (operación de restricción) o un fragmento mezclado (combinación de operaciones de proyecto y de restricción). Un fragmento se asigna a un sitio, aunque en ocasiones a múltiples sitios. Si el DBMS distribuido soporta copiado, puede asignarse un fragmento a múltiples sitios. En algunos DBMS distribuidos que soportan copiado, una copia de un fragmento se considera la copia primaria y las otras copias son secundarias. Sólo se garantiza que la copia primaria sea la actual.

La arquitectura de esquema para una DBMS distribuida con integración floja soporta más autonomía de sitios de base de datos locales, además de compartición de datos. Cada sitio

**FIGURA 17.14**  
Arquitectura de  
esquema para un  
DBMS distribuido  
firmemente integrado



**FIGURA 17.15**  
**Arquitectura de esquema para un DBMS distribuido con integración floja**



contiene los tres niveles de esquema tradicionales, como se muestra en la figura 17.15. Para soportar la compartición de datos, el DBMS distribuido proporciona un esquema de mapeo local para cada sitio. Los esquemas de mapeo local describen los datos exportables en un sitio y proporcionan reglas de conversión para traducir los datos de un formato local a uno global. El esquema conceptual global muestra todos los tipos de datos y relaciones que se pueden usar en solicitudes globales. Algunos DBMSs distribuidos no tienen un esquema conceptual global. En vez de ello, los esquemas externos globales proporcionan visiones de datos compartidos en un formato común.

Puede haber muchas diferencias entre los formatos de datos locales. Los sitios locales pueden usar diferentes DBMSs, cada uno con un conjunto diferente de tipos de datos. Los modelos de datos de los DBMS locales pueden ser diferentes, en especial si se integran sistemas heredados. Los sistemas heredados pueden usar interfaces de archivo y modelos de datos navegacionales (red y jerarquía) que no soportan SQL. Puede haber muchas diferencias incluso si los sitios locales soportan un estándar SQL común, tales como diferentes tipos de datos, escalas, unidades de medida y códigos. Los esquemas de mapeo local resuelven estas diferencias al proporcionar reglas de conversión que transforman los datos de un formato local a un formato global.

Las arquitecturas firmemente integradas y con integración floja representan dos posibilidades extremas. Entre estas dos arquitecturas se ha propuesto e implementado muchas variaciones. Por ejemplo, para proporcionar autonomía local adicional con mayor eficiencia para solicitudes globales, un sistema con integración floja puede requerir que todos los sitios locales soporten una interfaz SQL común. También pueden combinarse los enfoques firmemente integrado y con integración floja. Las redes de bases de datos distribuidas firmemente integradas pueden tener integración floja para compartir datos selectivos en solicitudes globales. En este caso, el DBMS distribuido con integración floja actúa como puerta de enlace entre las bases de datos distribuidas firmemente integradas.

## 17.5 Transparencia para procesamiento de bases de datos distribuidas

Recuerde del capítulo 15 que transparencia se refiere a la visibilidad de los detalles internos de un servicio (visible u oculto). En el procesamiento de transacciones, los servicios de concurrencia y recuperación son transparentes u ocultos para los usuarios de la base de datos. El procesamiento de bases de datos paralelas enfatiza la transparencia. En el procesamiento de bases de datos distribuidas, la transparencia se relaciona con la independencia de datos. Si la distribución de la base de datos es transparente, los usuarios pueden escribir consultas sin conocimiento de su distribución. Además, los cambios en la distribución no causarán cambios a las consultas y transacciones existentes. Si la distribución de base de datos no es transparente, los usuarios deben referirse a ciertos detalles de distribución en las consultas, y los cambios de distribución pueden conducir a cambios en las consultas existentes.

Esta sección describe niveles comunes de transparencia y proporciona ejemplos de formulación de consultas con cada nivel. Antes de discutir los niveles de transparencia, se presenta un ejemplo motivacional.

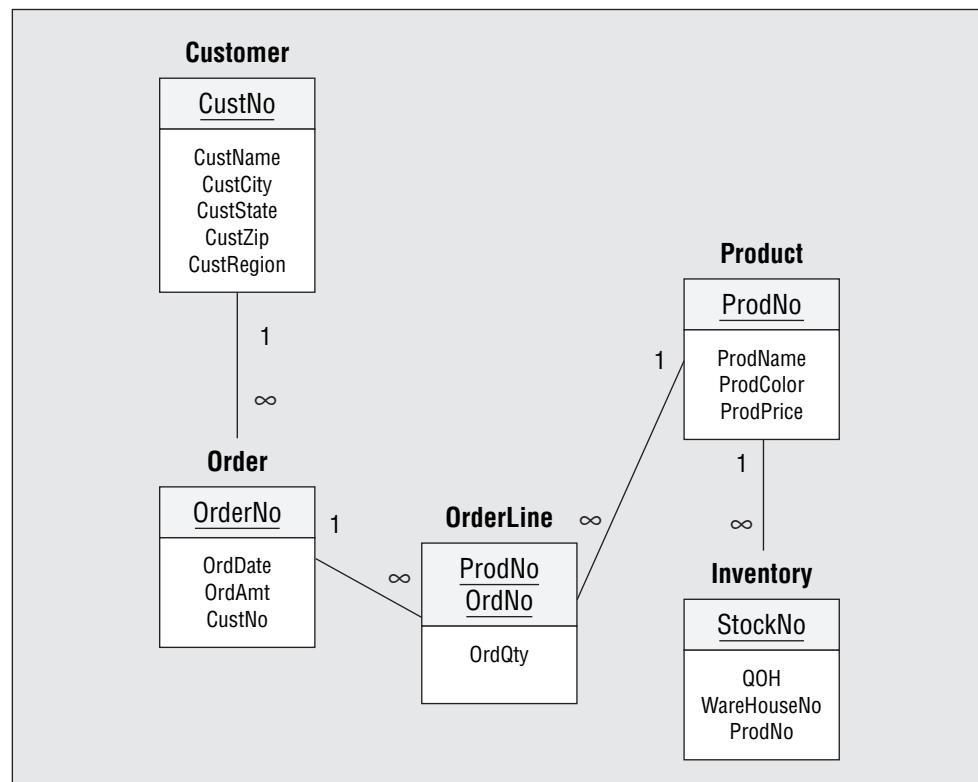
### 17.5.1 Ejemplo motivacional

A continuación se proporcionan más detalles acerca de la base de datos de solicitud de pedido con el fin de mostrar los niveles de transparencia. La base de datos de solicitud de pedidos contiene cinco tablas, como se muestra en el diagrama de relación de la figura 17.16. Debe suponer que los clientes se ubican en dos regiones (East y West) y los productos se almacenan en dos almacenes (1: Denver, 2: Seattle).

Una colección de fragmentos puede definirse utilizando la columna de región de cliente, como se muestra en la tabla 17.3. El fragmento *Western-Customers* consiste en clientes con una región igual a “West”. Hay dos fragmentos relacionados: el fragmento *Western-Orders*, que consiste en los pedidos de los clientes del oeste, y el fragmento *Western-OrderLines*, que consiste en las líneas de pedido equivalentes a pedidos del oeste. Para las filas que involucran clientes del este se definen fragmentos similares.

**FIGURAS 17.16**

Diagrama relacional para la base de datos de solicitud de pedidos



**TABLA 17.3**  
Fragmentos con base en el campo *CustRegion*<sup>1</sup>

```

CREATE FRAGMENT Western-Customers AS
    SELECT * FROM Customer WHERE CustRegion = 'West'

CREATE FRAGMENT Western-Orders AS
    SELECT Order.* FROM Order, Customer
        WHERE Order.CustNo = Customer.CustNo AND CustRegion = 'West'

CREATE FRAGMENT Western-OrderLines AS
    SELECT OrderLine.* FROM Customer, OrderLine, Order
        WHERE OrderLine.OldNo = Order.OldNo
            AND Order.CustNo = Customer.CustNo AND CustRegion = 'West'

CREATE FRAGMENT Eastern-Customers AS
    SELECT * FROM Customer WHERE CustRegion = 'East'

CREATE FRAGMENT Eastern-Orders AS
    SELECT Order.* FROM Order, Customer
        WHERE Order.CustNo = Customer.CustNo AND CustRegion = 'East'

CREATE FRAGMENT Eastern-OrderLines AS
    SELECT OrderLine.* FROM Customer, OrderLine, Order
        WHERE OrderLine.OldNo = Order.OldNo
            AND Order.CustNo = Customer.CustNo AND CustRegion = 'East'

```

**TABLA 17.4**  
Fragmentos con base en el campo *WarehouseNo*

```

CREATE FRAGMENT Denver-Inventory AS
    SELECT * FROM Inventory WHERE WareHouseNo = 1

CREATE FRAGMENT Seattle-Inventory AS
    SELECT * FROM Inventory WHERE WareHouseNo = 2

```

**semiunión**  
operador del álgebra relacional especialmente útil para el procesamiento de bases de datos distribuidas. Una semiunión es la mitad de una combinación: las filas de una tabla que empatan con al menos una fila de otra tabla. En el resultado aparecen sólo las filas de la primera tabla.

Los fragmentos de pedido y línea de pedido se derivan a partir de un fragmento de cliente con el uso del operador de semiunión: una semiunión es la mitad de una combinación: las filas de una tabla que empatan con las filas de otra tabla. Por ejemplo, una operación de semiunión define el fragmento *Western-Orders* como las filas de la tabla *Order* que empatan con las filas de cliente con una región de “West”. Al fragmento definido con una operación de semiunión a veces se le llama fragmento horizontal derivado. El operador de semiunión es importante para definir fragmentos, ya que algunos fragmentos deben tener filas relacionadas con otros fragmentos.

Los fragmentos de almacén se definen con el uso de la columna *WareHouseNo*, como se muestra en la tabla 17.4. En las definiciones de fragmento, se supone que el almacén número 1 se ubica en Denver y el almacén número 2, en Seattle. La tabla *Product* no está fragmentada porque toda la tabla se copia en múltiples sitios.

La fragmentación puede ser más compleja que lo descrito en la base de datos de solicitud de pedidos. Puede haber muchos fragmentos adicionales para acomodar una estructura empresarial. Por ejemplo, si hay centros y bodegas de procesamiento de cliente adicionales, pueden definirse fragmentos adicionales. Además, los fragmentos verticales pueden definirse como operaciones de proyección adicional a los fragmentos horizontales con el uso de las operaciones de restricción y semiunión. Incluso puede definirse un fragmento como una combinación de operaciones de proyección, restricción y semiunión. La única limitación es que los fragmentos no deben estar combinados. La falta de combinación significa que los fragmentos horizontales no contienen filas comunes y que los fragmentos verticales no contienen columnas comunes, excepto por la llave primaria.

<sup>1</sup> La sintaxis de la tabla 17.3 no es una sintaxis SQL oficial.

**TABLA 17.5**  
**Asignación de fragmentos de la base de datos de solicitud de pedidos**

Fragmentos	Sitio
Western-Customers, Western-Orders, Western-OrderLines	Boise
Eastern-Customers, Eastern-Orders, Eastern-OrderLines	Tulsa
Denver-Inventory, Product	Denver
Seattle-Inventory, Product	Seattle

**TABLA 17.6**  
**Solicitudes representativas con el uso de la base de datos de solicitud de pedidos**

<b>Encontrar pedido</b>
SELECT * FROM Order, Customer WHERE Order.CustNo = \$X AND Order.CustNo = Customer.CustNo
<b>Encontrar disponibilidad de producto</b>
SELECT * FROM Inventory WHERE ProdNo = \$X
<b>Actualizar inventario</b>
UPDATE Inventory SET QOH = QOH - 1 WHERE ProdNo = \$X AND WareHouseNo = \$Y
<b>Mover cliente</b>
UPDATE Customer SET CustRegion = \$X WHERE CustNo = \$Y

Una vez definidos los fragmentos, son asignados a los sitios. En ocasiones, los fragmentos se definen con base en el lugar donde deben asignarse. La asignación de los fragmentos de solicitud de pedidos sigue este enfoque, como se muestra en la tabla 17.5. El sitio Boise contiene los fragmentos de los clientes del oeste, mientras que el sitio Tulsa contiene los fragmentos de los clientes del este. De igual modo, los fragmentos de inventario se dividen entre los sitios Denver y Seattle. La tabla *Product* se copia en los sitios Denver y Seattle, porque cada almacén tiene cada producto.

En la práctica, el diseño y la asignación de fragmentos es mucho más difícil que lo mostrado aquí. Diseñar y asignar fragmentos es similar a la selección de índice. Son necesarios datos sobre la frecuencia de consultas, frecuencia de valores de parámetro en consultas y comportamiento del optimizador de consulta global. Además, son necesarios datos acerca de la frecuencia de los sitios de origen de cada consulta. El sitio de origen para una consulta es aquel donde la consulta se almacena o desde donde se emite. Al igual que en la selección de índice, los modelos y herramientas de optimización pueden auxiliar a la toma de decisiones acerca del diseño y asignación de fragmentos. Los detalles de los modelos de optimización y las herramientas están más allá del ámbito de este libro. Las referencias al final del capítulo ofrecen detalles acerca del diseño y asignación de fragmentos.

### 17.5.2 Transparencia de fragmentación

**transparencia de fragmentación**  
nivel de independencia en DBMSs distribuidos, en la que la consulta puede formularse sin conocimiento de los fragmentos.

La transparencia de fragmentación proporciona el mayor nivel de independencia de datos. Los usuarios formulan consultas y transacciones sin conocimiento de los fragmentos, ubicaciones o formatos locales. Si los fragmentos, ubicaciones o formatos locales cambian, las consultas y transacciones no se afectan. En esencia, los usuarios perciben la base de datos distribuida como una base de datos centralizada. La transparencia de fragmentación implica el menor trabajo para los usuarios, pero el mayor trabajo para el DBMS distribuido. El procesamiento de bases de datos paralelas en las arquitecturas nada compartido involucra transparencia de fragmentación.

Para contrastar los niveles de transparencia, la tabla 17.6 cita algunas consultas y transacciones representativas que usan la base de datos de solicitud de pedidos. En estas consultas, en lugar de valores individuales se usa los parámetros \$X y \$Y. Con la transparencia de fragmentación, las consultas y transacciones pueden emitirse sin cambio, sin importar la fragmentación de la base de datos.

### 17.5.3 Transparencia de ubicación

#### transparencia de ubicación

nivel de independencia en DBMS distribuidos en la que las consultas pueden formularse sin conocimiento de las ubicaciones. Sin embargo, es necesario el conocimiento de los fragmentos.

La transparencia de ubicación proporciona un menor nivel de independencia de datos que la transparencia de fragmentación. Los usuarios necesitan referirse a fragmentos al formular consultas y transacciones. Sin embargo, no es necesario el conocimiento de las ubicaciones y formatos locales. Aun cuando no es necesario el conocimiento del sitio, los usuarios están indirectamente conscientes de la distribución de una base de datos porque muchos fragmentos se asignan a un solo sitio. Los usuarios pueden hacer una asociación entre fragmentos y sitios.

La transparencia de ubicación implica más trabajo al formular solicitudes, como se muestra en la tabla 17.7. En las consultas “encontrar”, el operador unión recopila filas de todos los fragmentos. La consulta Actualizar inventario implica aproximadamente la misma cantidad de codificación. El usuario sustituye el nombre de un fragmento en lugar de la condición en *WarehouseNo*, porque esta condición define el fragmento.

**TABLA 17.7**  
Solicitudes escritas con transparencia de ubicación

<b>Encontrar pedido</b>
<pre>SELECT * FROM Western-Orders, Western-Customers       WHERE Western-Orders.CustNo = \$X             AND Western-Orders.CustNo = Western-Customers.CustNo UNION SELECT * FROM Eastern-Orders, Eastern-Customers       WHERE Eastern-Orders.CustNo = \$X             AND Eastern-Orders.Custno = Eastern-Customers.CustNo</pre>
<b>Encontrar disponibilidad de producto</b>
<pre>SELECT * FROM Denver-Inventory       WHERE ProdNo = \$X UNION SELECT * FROM Seattle-Inventory       WHERE ProdNo = \$X</pre>
<b>Actualizar inventario (Denver)</b>
<pre>UPDATE Denver-Inventory SET QOH = QOH - 1       WHERE ProdNo = \$X</pre>
<b>Mover cliente (West a East)</b>
<pre>SELECT CustName, CustCity, CustState, CustZip       INTO \$CustName, \$CustCity, \$CustState, \$CustZip       FROM Western-Customers WHERE CustNo = \$Y  INSERT INTO Eastern-Customers (CustNo, CustName, CustCity, CustState, CustZip, CustRegion)       VALUES (\$Y, \$CustName, \$CustCity, \$CustState, \$CustZip, 'East')  INSERT INTO Eastern-Orders       SELECT * FROM Western-Orders WHERE CustNo = \$Y  INSERT INTO Eastern-OrderLines       SELECT * FROM Western-OrderLines       WHERE OrdNo IN             (SELECT OrdNo FROM Western-Orders WHERE CustNo = \$Y)  DELETE FROM Western-OrderLines       WHERE OrdNo IN             (SELECT OrdNo FROM Western-Orders WHERE CustNo = \$Y)  DELETE Western-Orders WHERE CustNo = \$Y  DELETE Western-Customers WHERE CustNo = \$Y</pre>

En la solicitud Mover cliente, es necesario mucho más código. No puede usarse una operación actualizar porque la columna a actualizar define el fragmento. En vez de ello, deben insertarse filas en los nuevos fragmentos y borrar las de los fragmentos antiguos. Para el fragmento de clientes, el enunciado SELECT . . . INTO almacena valores de campo en variables que se usan en el siguiente enunciado INSERT. Los borrados se realizan en el orden establecido si las filas referenciadas deben borrarse al final. Si el borrado es en cascada, sólo es necesario un enunciado DELETE en el fragmento *Western-Customers*.

Los enunciados SQL para las primeras dos solicitudes no revelan el número de operaciones unión que puedan requerirse. Con dos fragmentos, se necesita una operación unión. Sin embargo, con  $n$  fragmentos, son necesarias  $n-1$  operaciones unión.

En cierta medida, las vistas pueden proteger a los usuarios de algunos de los detalles de fragmento. Por ejemplo, al usar una vista definida con operaciones de unión, se obviaría la necesidad de escribir las operaciones de unión en la consulta. Sin embargo, las vistas pueden no simplificar los enunciados de manipulación. Si el DBMS no soporta transparencia de fragmentación, es poco probable que las vistas actualizables puedan abarcar sitios. Por lo tanto, el usuario todavía tendría que escribir los enunciados SQL para la solicitud Mover cliente.

#### 17.5.4 Transparencia de mapeo local

**transparencia de mapeo local**  
nivel de independencia en DBMS distribuidos en donde las consultas pueden formularse sin conocimiento de formatos locales. Sin embargo, se necesita el conocimiento de los fragmentos y las asignaciones de fragmento (ubicaciones).

La transparencia de mapeo local proporciona un menor nivel de independencia de datos que la transparencia de ubicación. Los usuarios necesitan hacer referencia a fragmentos en sitios al formular consultas y transacciones. Sin embargo, no es necesario el conocimiento de los formatos locales. Si los sitios difieren en formatos, como en las bases de datos distribuidas con integración floja, la transparencia de mapeo local todavía libera al usuario de un trabajo considerable.

La transparencia de ubicación puede no implicar mucho esfuerzo de codificación adicional del que se muestra en la tabla 17.8. Los únicos cambios entre las tablas 17.7 y 17.8 son la adición de los nombres de sitio en la tabla 17.8. Si los fragmentos se copian, se necesita codificación adicional en las transacciones. Por ejemplo, si se agrega un nuevo producto con transparencia de mapeo local, se necesitan dos enunciados INSERT (uno para cada sitio). Con transparencia de ubicación, sólo es necesario un enunciado INSERT. La cantidad de codificación adicional depende de la cantidad de copiado.

A partir de lo expuesto en esta sección, usted puede suponer falsamente que se prefiere la transparencia de fragmentación a los otros niveles de transparencia. La transparencia de fragmentación proporciona el nivel más alto de independencia de datos, pero es la más compleja de implementar. Para el procesamiento de bases de datos paralelas en arquitecturas de nada compartido, la transparencia de fragmentación es una característica clave. Para bases de datos distribuidas, la transparencia de fragmentación entra en conflicto con la meta de la autonomía del sitio. La propiedad de datos implica atención de los usuarios cuando combinan datos locales y remotos. Además, la transparencia de fragmentación puede alentar consumo excesivo de recursos porque los usuarios no perciben el procesamiento subyacente de bases de datos distribuidas. Con la transparencia de ubicación y de mapeo local, los usuarios perciben el procesamiento subyacente de base de datos distribuidas al menos en cierta medida. La cantidad y complejidad del procesamiento de bases de datos distribuidas puede ser considerable, como se describe en la sección 17.6.

#### 17.5.5 Transparencia en bases de datos distribuidas de Oracle

Oracle 10g soporta bases de datos distribuidas homogéneas y heterogéneas. En el caso homogéneo, cada sitio contiene una base de datos Oracle gestionada por separado. El requisito de gestión separada proporciona autonomía para cada sitio participante. Las bases de datos individuales pueden utilizar cualquier versión Oracle soportada, aunque la funcionalidad en las solicitudes locales está limitada a la base de datos de la versión más baja. Oracle soporta replicación en bases de datos distribuidas mediante sitios maestros designados y procesamiento asíncrono en sitios secundarios. Las bases de datos no Oracle también pueden participar en solicitudes globales que

**TABLA 17.8**  
**Solicitudes escritas**  
**con transparencia de**  
**mapeo local**

<b>Encontrar pedido</b>
SELECT * FROM Western-Orders@Boise, Western-Customers@Boise WHERE Western-Orders@Boise.CustNo = \$X AND Western-Orders@Boise.CustNo = Western-Customers@Boise.CustNo
UNION
SELECT * FROM Eastern-Orders@Tulsa, Eastern-Customers@Tulsa WHERE Eastern-Orders@Tulsa.CustNo = \$X AND Eastern-Orders@Tulsa.CustNo = Eastern-Customers@Tulsa.CustNo
<b>Encontrar disponibilidad de producto</b>
SELECT * FROM Denver-Inventory@Denver WHERE ProdNo = \$X
UNION
SELECT * FROM Seattle-Inventory@Seattle WHERE ProdNo = \$X
<b>Actualizar inventario (Denver)</b>
UPDATE Denver-Inventory@Denver SET QOH = QOH - 1 WHERE ProdNo = \$X
<b>Mover cliente (West a East)</b>
SELECT CustName, CustCity, CustState, CustZip INTO \$CustName, \$CustCity, \$CustState, \$CustZip FROM Western-Customers@Boise WHERE CustNo = \$Y
INSERT INTO Eastern-Customers@Tulsa (CustNo, CustName, CustCity, CustState, CustZip, CustRegion) VALUES (\$Y, \$CustName, \$CustCity, \$CustState, \$CustZip, 'East')
INSERT INTO Eastern-Orders@Tulsa SELECT * FROM Western-Orders@Boise WHERE CustNo = \$Y
INSERT INTO Eastern-OrderLines@Tulsa SELECT * FROM Western-OrderLines@Boise WHERE OrdNo IN (SELECT OrdNo FROM Western-Orders@Boise WHERE CustNo = \$Y)
DELETE FROM Western-OrderLines@Boise WHERE OrdNo IN (SELECT OrdNo FROM Western-Orders@Boise WHERE CustNo = \$Y)
DELETE Western-Orders@Boise WHERE CustNo = \$Y
DELETE Western-Customers@Boise WHERE CustNo = \$Y

usan servicios heterogéneos y agentes de entrada. Esta sección proporciona detalles acerca de la transparencia en bases de datos distribuidas puras (no copiadas).

Los vínculos a bases de datos son un concepto clave para bases de datos distribuidas Oracle. Un vínculo a una base de datos proporciona conexión de una vía desde una base de datos local hasta una base de datos remota. Una base de datos local es la base de datos en la que se conecta un usuario. Una base de datos remota es otra base de datos en la que un usuario desea acceder en una solicitud global. Los vínculos a bases de datos permiten al usuario acceder a los objetos de

**TABLA 17.9**  
**Enunciados Oracle**  
**para solicitud global**

<b>Crear vínculo</b>
CREATE DATABASE LINK boise.acme.com CONNECT TO clerk1 IDENTIFIED BY clerk1
<b>Encontrar pedido (uso del nombre del vínculo)</b>
SELECT * FROM Order@boise.acme.com WO, Customer@boise.acme.com WC WHERE WO.CustNo = 1111111 AND WO.CustNo = WC.CustNo UNION SELECT * FROM Order, Customer WHERE Order.CustNo = 1111111 AND Order.CustNo = Customer.CustNo
<b>Crear sinónimo</b>
CREATE PUBLIC SYNONYM BoiseOrder FOR order@boise.acme.com; CREATE PUBLIC SYNONYM BoiseCustomer FOR customer@boise.acme.com;
<b>Encontrar pedido (uso del nombre del vínculo)</b>
SELECT * FROM BoiseOrder WO, BoiseCustomer WC WHERE WO.CustNo = 1111111 AND WO.CustNo = WC.CustNo UNION SELECT * FROM Order, Customer WHERE Order.CustNo = 1111111 AND Order.CustNo = Customer.CustNo

otro usuario en una base de datos remota sin tener una cuenta en el sitio remoto. Cuando se usa un vínculo a bases de datos, el usuario remoto está limitado por el conjunto de privilegios del objeto propietario.

La tabla 17.9 demuestra los enunciados Oracle para crear vínculos y sinónimos, así como su uso en una consulta global. El primer enunciado crea un vínculo fijo a la base de datos “boise.acme.com” a través del usuario remoto “clerk1”. El siguiente enunciado usa el vínculo para acceder a la tabla remota *Order*. Se supone que el usuario actual está conectado a la base de datos de Tulsa y usa lígale vínculo para acceder a la base de datos Boise. En la cláusula FROM, los nombres de tabla no calificados en ambos sitios son los mismos (*Order* y *Customer*). Los enunciados CREATE SYNONYM crean alias para tablas remotas con el uso de los nombres de la tabla remota y los nombres del vínculo. Los enunciados SELECT pueden usar el sinónimo en lugar de los nombres de tabla y el vínculo.

Como muestra la tabla 17.9, los vínculos a base de datos proporcionan transparencia de mapeo local a datos remotos. Para crear un vínculo, un usuario debe conocer el nombre de la base de datos global. El nombre de una base de datos global usualmente contiene información acerca de la estructura y las ubicaciones de negocios de una organización. Para usar un vínculo, un usuario debe conocer los nombres y detalles del objeto remoto. La transparencia de mapeo local es consistente con cierto énfasis en la autonomía del sitio. Oracle permite más transparencia (transparencia de ubicación) en acceso a datos remotos a través del uso de sinónimos y vistas.

La Guía de Administradores de Bases de Datos de Oracle 10g proporciona más detalles acerca de la transparencia de bases de datos distribuidas. Los detalles incluyen muchos ámbitos de vínculo (público, privado y global), muchos tipos de conexiones remotas y administración de roles y privilegios para acceso a bases de datos remotas.

## 17.6 Procesamiento de bases de datos distribuidas

---

Así como los datos distribuidos pueden agregar complejidad para la formulación de consultas, también añaden considerable complejidad al procesamiento de consultas y transacciones. El procesamiento de bases de datos distribuidas implica movimiento de datos, procesamiento remoto y coordinación del sitio que están ausentes del procesamiento de bases de datos centralizadas. Aunque los detalles del procesamiento de bases de datos distribuidas pueden ocultarse a los programadores y usuarios, a veces no es posible ocultar las implicaciones de rendimiento. Esta sección presenta detalles acerca del procesamiento de consultas distribuidas y del procesamiento de transacciones distribuidas a fin de tomar conciencia acerca de las complejidades que pueden afectar el rendimiento.

### 17.6.1 Procesamiento de consulta distribuida

El procesamiento de consulta distribuida es más complejo que el procesamiento de consulta centralizada debido a muchas razones. El procesamiento de consulta distribuida implica optimización tanto local (intra-sitio) como global (inter-sitio). La optimización global implica decisiones de movimiento de datos y de selección de sitio, ausentes en el procesamiento de consulta centralizada. Por ejemplo, para realizar una combinación de fragmentos distribuidos, puede moverse un fragmento, ambos fragmentos pueden moverse hacia un tercer sitio o sólo pueden moverse los valores de combinación de un fragmento. Si los fragmentos se copian, entonces debe elegirse un sitio para cada fragmento.

Muchas de las complejidades del procesamiento de consulta distribuida también existen para las bases de datos paralelas con arquitecturas nada compartidas. La principal diferencia es la comunicación en redes mucho más rápida y más confiable que las que son usadas en el procesamiento de bases de datos paralelas.

El procesamiento de consultas distribuidas también es más complejo porque existen múltiples objetivos de optimización. En un entorno centralizado, minimizar el uso de recursos (entrada-salida y procesamiento) es consistente con la reducción del tiempo de respuesta. En un entorno distribuido, minimizar los recursos puede entrar en conflicto con la reducción del tiempo de respuesta, debido a las oportunidades de procesamiento paralelo. El procesamiento paralelo puede reducir el tiempo de respuesta para aumentar la cantidad global de recursos consumidos (entrada-salida, procesamiento y comunicación). Además, la ponderación de los costos de comunicación contra los costos locales (entrada-salida y procesamiento) depende de las características de la red. Para redes de área amplia, los costos de comunicación pueden dominar sobre los costos locales. Para redes de área local, los costos de comunicación están más equitativamente ponderados con los costos locales.

La creciente complejidad hace que la optimización de las consultas distribuidas sea incluso más importante que la optimización de las consultas centralizadas. Debido a que el procesamiento de consulta distribuido implica tanto optimización local como global, existen muchos más planes de acceso posibles para una consulta distribuida que para su correspondiente consulta centralizada. La discrepancia en rendimiento entre planes de acceso distribuido puede ser muy grande. La elección de un mal plan de acceso puede conducir a un rendimiento extremadamente pobre. Además, los planes de acceso distribuidos a veces necesitan ajustarse para condiciones del sitio. Si un sitio no está disponible o está sobrecargado, un plan de acceso distribuido deberá elegir dinámicamente otro sitio. Por ende, parte del proceso de optimización puede requerir que se ejecute dinámicamente (durante el tiempo de ejecución) en vez de estáticamente (durante el tiempo de compilación).

Para mostrar la importancia de la optimización de consulta distribuida, se presentan los planes de acceso para una consulta muestra. Para simplificar la presentación, se usa una red pública con tiempos de comunicación relativamente lentos. Sólo se muestran los tiempos de comunicación ( $CT$ ) para cada plan de acceso. El tiempo de comunicación consiste en un retardo de mensaje fijo ( $MD$ ) y un tiempo de transmisión variable ( $TT$ ). Cada registro se transmite como un mensaje separado.

**TABLA 17.10**  
**Estadísticas para la consulta y la red**

La longitud del registro es 1 000 bits para cada tabla.
El cliente tiene 5 pedidos en el rango de fecha especificado.
Cada pedido contiene un promedio de 5 productos.
El cliente tiene 3 pedidos en el rango de fecha y color especificados.
Hay 200 productos rojos.
Hay 10 000 pedidos, 50 000 líneas de pedido y 1 000 productos en los fragmentos.
La asignación de fragmento está dada en la tabla 17.6.
El retardo por mensaje es de 0.1 segundo.
La tasa de datos es 1 000 000 bits por segundo.

$$CT = MD + TT$$

$MD$  = Número de mensajes \* Retardo por mensaje

$TT$  = Número de bits/Tasa de datos

*Consulta global:* Enlista el número de pedido, fecha de pedido, número de producto, nombre de producto, precio de producto y cantidad de pedido para pedidos del este con un número de cliente especificado, rango de fecha y color de producto. La tabla 17.10 menciona las estadísticas para la consulta y la red.

```
SELECT * EO.OrdNo, OrdDate, P.ProdNo, QtyOrd, ProdName, ProdPrice
  FROM Eastern-Orders EO, Eastern-Orderlines EOL, Product P
 WHERE EO.CustNo = $X AND EO.OrdNo = EOL.OrdNo AND P.Color = 'Red'
       AND EOL.ProdNo = P.ProdNo and OrdDate BETWEEN $Y AND $Z
```

1. Mover la tabla *Product* al sitio Tulsa, de donde procede la consulta.

$$CT = 1000 * 0.1 + (1000 * 1000) / 1000000 = 101 \text{ segundos}$$

2. Restringe la tabla *Product* al sitio Denver. Luego mueve el resultado a Tulsa, donde se procesa el resto de la consulta.

$$CT = 200 * 0.1 + (200 * 1000) / 1000000 = 20.2 \text{ segundos}$$

3. Realiza unión y restricciones de los fragmentos *Eastern-Orders* y *Eastern-OrderLines* en el sitio Tulsa. Mueve el resultado al sitio Denver para combinarlo con la tabla *Product*.

$$CT = 25 * 0.1 + (25 * 2000) / 1000000 = 2.55 \text{ segundos}$$

4. Restringe la tabla *Product* en el sitio Denver. Mueve sólo los números de producto resultantes (32 bits) a Tulsa. Realiza uniones y restricciones en Tulsa. Mueve los resultados de vuelta a Denver para combinarlos con la tabla *Product*.

$$CT(\text{Denver a Tulsa}) = 200 * 0.1 + (200 * 32) / 1000000 = 20.0064 \text{ segundos}$$

$$CT(\text{Tulsa a Denver}) = 15 * 0.1 + (15 * 2000) / 1000000 = 1.53 \text{ segundos}$$

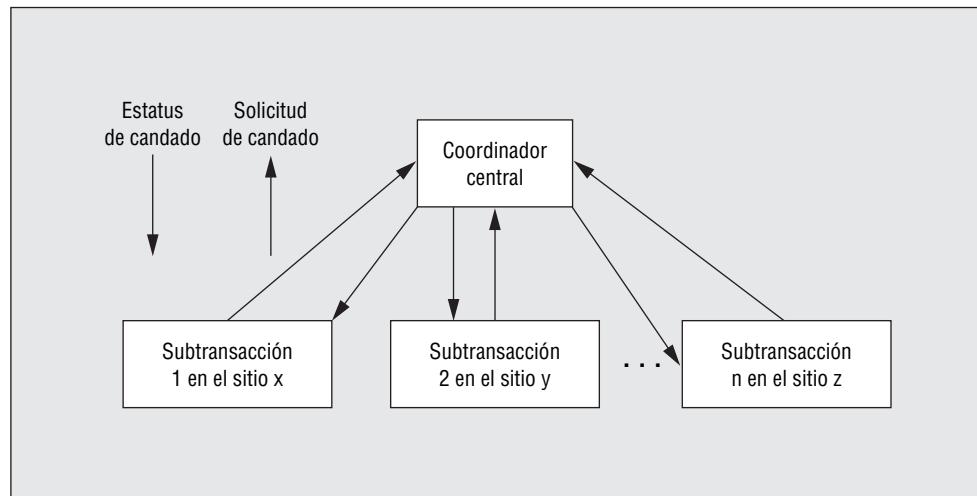
$$CT = CT(\text{Denver a Tulsa}) + CT(\text{Tulsa a Denver}) = 21.5364 \text{ segundos}$$

Estos planes de acceso muestran una amplia variación en los tiempos de comunicación. Incluso se mostraría más variación si los fragmentos de pedido se moviesen de Tulsa a Denver. El tercer plan de acceso domina a los otros debido a su bajo retraso de mensaje. Para determinar el mejor plan de acceso se necesitan análisis adicionales de los costos de procesamiento locales.

## 17.6.2 Procesamiento de transacción distribuida

El procesamiento de transacción distribuida sigue los principios descritos en el capítulo 15. Las transacciones obedecen las propiedades ACID y los DBMS distribuidos proporcionan concurrencia y transparencia de recuperación. Sin embargo, un entorno distribuido hace más difícil la implementación de los principios. Los sitios que operan independientemente deben estar coordinados. Además, existen nuevos tipos de fallas debido a la red de comunicación. Para lidiar con estas complejidades son necesarios nuevos protocolos. Esta sección presenta una introducción

**TABLA 17.17**  
**Control de con-**  
**currencia**  
**centralizada**



a los problemas y soluciones del control de concurrencia distribuida y del procesamiento de compromiso.

#### *Control de concurrencia distribuida*

El control de concurrencia distribuida puede implicar más gastos generales que el control de concurrencia centralizada, pues los sitios locales deben coordinarse a través de mensajes sobre una red de comunicación. El esquema más simple implica coordinación centralizada, como se muestra en la figura 17.17. Al comienzo de una transacción, se elige el sitio de coordinación y la transacción se divide en subtransacciones realizadas en otros sitios. Cada sitio que alberga una transacción emite solicitudes de candado y liberación al sitio coordinador mediante el uso de reglas normales de candado en dos fases.

La coordinación centralizada implica la menor cantidad de mensajes y la detección más simple de bloqueo. Sin embargo, la confiabilidad en un coordinador centralizado puede hacer que el procesamiento de transacción sea menos confiable. Para aliviar la confianza en un sitio centralizado, el administrador de candados puede distribuirse entre sitios. El precio por mayor confiabilidad es más gastos generales por mensaje y detección de bloqueo más compleja. El número de mensajes puede duplicarse en el esquema de coordinación distribuida en comparación con el esquema de coordinación centralizada.

Los datos copiados implican un problema tanto en la coordinación centralizada como en la distribuida. Actualizar los datos copiados implica gastos generales adicionales debido a que se debe obtener un candado de escritura en todas las copias antes de que cualquier copia se actualice. Obtener candados de escritura en múltiples copias puede causar retardos e incluso regresiones si una copia no está disponible.

Para reducir la saturación con múltiples copias de candado puede usarse el protocolo de copia primaria. En el protocolo de copia primaria, una copia de cada fragmento copiado se designa como la copia primaria, mientras que las otras copias son secundarias. Los candados de escritura son necesarios sólo para la copia primaria. Después de que una transacción actualiza la copia primaria, las actualizaciones se propagan a las copias secundarias. Sin embargo, las copias secundarias pueden no actualizarse sino hasta después de comprometer la transacción. El protocolo de copia primaria proporciona rendimiento mejorado pero al costo de copias secundarias no concurrentes. Debido a que la reducción de gastos generales con frecuencia es más importante que las copias secundarias actuales, muchos DBMS distribuidos usan el protocolo de copia primaria.

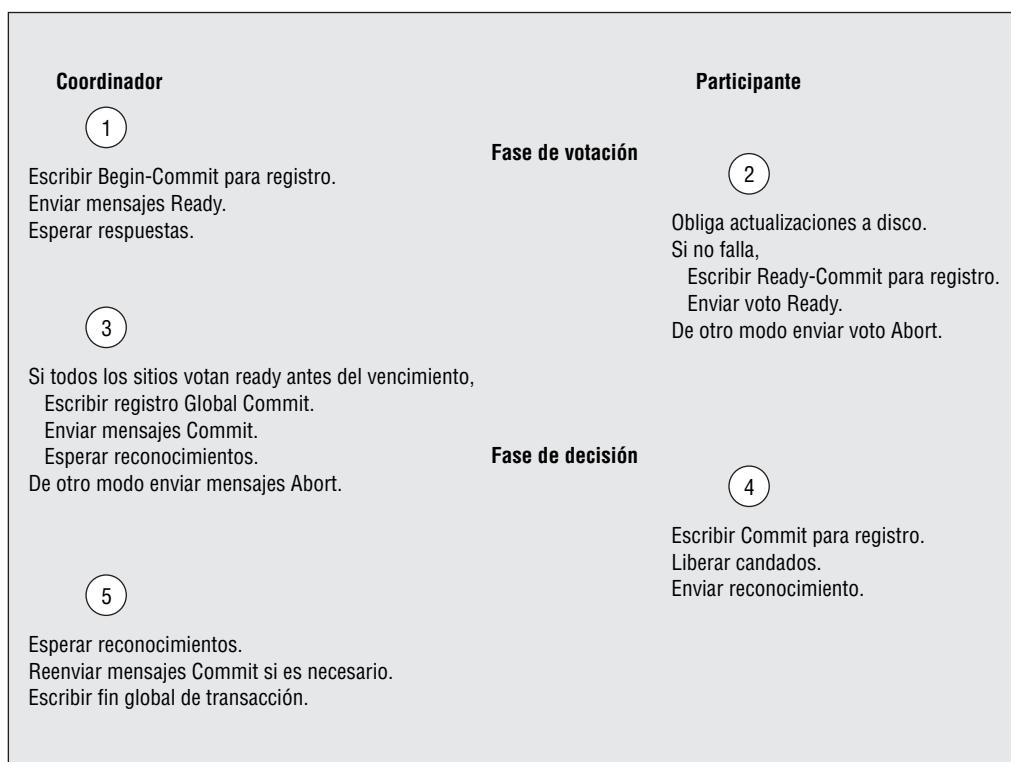
#### *Procesamiento de compromiso distribuido*

Los DBMS distribuidos deben enfrentar fallas de comunicación en vínculos y sitios, fallas que no afectan a los DBMS centralizados. La detección de fallas implica coordinación entre sitios.

#### **protocolo de copia primaria**

Protocolo para el control de concurrencia de transacciones distribuidas. Cada fragmento copiado está designado como la copia primaria o como la copia secundaria. Durante el procesamiento de transacción distribuida sólo la copia primaria tiene garantía de ser la actual al final de una transacción. Las actualizaciones pueden propagarse a copias secundarias después del fin de la transacción.

**FIGURA 17.18**  
**Procesamiento de compromiso en dos fases para coordinador y participantes**



Si un vínculo o sitio falla, debe abortarse cualquier transacción que involucre al sitio. Además, el sitio debe evitarse para futuras transacciones hasta que la falla se resuelva.

Las fallas pueden ser más complejas que sólo un sitio sencillo o un vínculo de comunicación. Muchos sitios y vínculos pueden fallar simultáneamente dejando una red particionada. En una red particionada las diferentes particiones (colecciones de sitios) no pueden comunicarse, aunque se comuniquen los sitios en la misma partición. El administrador de transacción debe asegurar que diferentes partes de una red particionada actúan al unísono. No es posible que los sitios en una partición decidan comprometer una transacción, sino que los sitios en otra partición deben decidir no comprometer una transacción. Todos los sitios deben comprometer o abortar.

El protocolo más extensamente conocido para procesamiento de compromiso distribuido es el protocolo de compromiso en dos fases.<sup>2</sup> Para cada transacción, se elige un sitio como el coordinador y la transacción se divide en subtransacciones realizadas en otros sitios participantes. El coordinador y los sitios participantes interactúan en una fase de votación y una fase de decisión. Al final de ambas fases, cada sitio participante actúa al unísono para comprometer o abortar su subtransacción.

Las fases de votación y decisión requieren acciones tanto del coordinador como de los sitios participantes, como se muestra en la figura 17.18. En la fase de decisión, el coordinador envía un mensaje a cada participante preguntando si está listo para comprometer. Antes de responder, cada participante obliga todas las actualizaciones a disco cuando termina el trabajo de la transacción local. Si no ocurre una falla, el participante escribe un registro de ready-commit y envía un voto ready al coordinador. En este punto, el participante tiene un estatus incierto porque el coordinador puede solicitar más tarde al participante abortar.

**protocolo de compromiso en dos fases (2PC)**  
 regla para asegurar que las transacciones distribuidas son atómicas. 2PC usa una fase de votación y una de decisión para coordinar compromisos de transacciones locales.

<sup>2</sup> No confundir compromiso en dos fases con cierre en dos fases. El protocolo de compromiso en dos fases se usa sólo para procesamiento de transacción distribuida. El cierre en dos fases puede usarse para control de concurrencia centralizado y distribuido.

La fase de decisión comienza cuando el coordinador recibe votos de cada participante u ocurre un vencimiento. Si ocurre un vencimiento o al menos un participante envía un voto de aborto, el coordinador aborta toda la transacción mediante el envío de mensajes abort a cada participante. Entonces cada participante realiza una regresión a sus cambios.

Si todos los participantes votan ready, el coordinador escribe el registro de compromiso global y pide a cada participante comprometer sus subtransacciones. Cada participante escribe un registro de compromiso, libera los cierres y envía un reconocimiento al coordinador. Cuando el coordinador recibe el reconocimiento de todos los participantes, el coordinador escribe el registro de fin de transacción global. Si ocurre una falla en la fase de votación o en la de decisión, el coordinador envía un mensaje abort a todos los sitios participantes.

En la práctica, el protocolo de compromiso en dos fases presentado en la figura 17.18 sólo es el protocolo básico. Otras complicaciones, como la falla durante la recuperación y vencimientos, pueden complicar el protocolo. Además, las modificaciones al protocolo básico pueden reducir el número de mensajes necesarios para forzar al protocolo. Como estas extensiones están más allá del ámbito de este libro, debe consultar las referencias al final del capítulo para más detalles.

El protocolo de compromiso en dos fases puede usar un coordinador centralizado o uno distribuido. Las negociaciones son similares a la coordinación centralizada frente a la distribuida para el control de concurrencias. La coordinación centralizada es más simple que la coordinación distribuida, pero puede ser menos confiable.

El protocolo de compromiso en dos fases no maneja ningún tipo concebible de falla. Por ejemplo, el protocolo de compromiso en dos fases puede no funcionar de manera adecuada si los registros de log se pierden. No hay protocolo conocido que garantice que todos los sitios actúan al unísono para comprometer o abortar al presentarse fallas arbitrarias. Como el protocolo de compromiso en dos fases maneja eficientemente tipos comunes de fallas, se usa ampliamente en procesamiento de transacción distribuida.

## Reflexión final

Este capítulo describió la motivación, arquitecturas y servicios de DBMS que soportan procesamiento distribuido y datos distribuidos. Utilizar procesamiento y datos distribuidos puede mejorar significativamente los servicios DBMS, pero al costo de nuevos retos de diseño. El procesamiento distribuido puede mejorar escalabilidad, interoperabilidad, flexibilidad, disponibilidad y rendimiento, mientras que los datos distribuidos pueden mejorar control de datos, costos de comunicación y rendimiento del sistema. Para darse cuenta de estos beneficios, deben superarse retos significativos causados por la complejidad del procesamiento y por los datos distribuidos.

Elegir una arquitectura adecuada es una forma de gestionar la complejidad adicional. Este capítulo describió las arquitecturas cliente-servidor y las arquitecturas de bases de datos paralelas para utilizar procesamiento distribuido. Las arquitecturas en dos, tres y múltiples niveles proporcionan alternativas entre costo, complejidad y flexibilidad para dividir las tareas entre clientes y servidores. El procesamiento de bases de datos paralelas distribuye una tarea grande entre los recursos disponibles. También se describió la tecnología de bases de datos paralelas de Oracle e IBM para indicar la implementación de las arquitecturas disco con clúster y nada con clúster.

La última parte del capítulo describió arquitecturas y procesamiento para DBMS distribuidos. Las arquitecturas para DBMS distribuidos difieren en la integración de las bases de datos locales. Los sistemas firmemente integrados proporcionan servicios de consulta y transacción, pero requieren uniformidad en el DBMS local. Los sistemas con integración floja soportan compartición de datos entre una mezcla de DBMS modernos y heredados. Una parte importante de la arquitectura de datos es el nivel de independencia de datos. Este capítulo describió muchos niveles de independencia de datos que difieren por el nivel de conocimiento requerido de distribución de datos para formular solicitudes globales. La presentación conceptual se complementó con ejemplos de transparencia en bases de datos distribuidas de Oracle. Este capítulo describió

el procesamiento de consulta distribuido y el procesamiento de transacción distribuido para proporcionar una comprensión introductoria a la complejidad del procesamiento de bases de datos distribuidas. Ambos servicios implican complejos conflictos que no se presentan en los DBMS centralizados.

---

## Revisión de conceptos

- Motivaciones para procesamiento cliente-servidor: escalabilidad, interoperabilidad y flexibilidad.
- Motivaciones para procesamiento de bases de datos paralelas: escalamiento, aceleración, alta disponibilidad, escalabilidad predecible.
- Motivaciones para datos distribuidos: mayor control local, reducción de costos de comunicación y rendimiento mejorado.
- Conflictos de diseño en procesamiento distribuido: división de procesamiento y gestión de proceso.
- Tipos de código para distribuir entre un cliente y un servidor.
- Tareas de administración de procesos realizadas por middleware de base de datos.
- Diferencias entre monitores de procesamiento de transacción, middleware orientado a mensaje, middleware de acceso a datos y agente de solicitud de objeto.
- Características de arquitecturas en dos, tres y múltiples niveles y servicios web.
- Características de arquitecturas de bases de datos paralelas: disco con clúster y nada con clúster.
- Problemas de procesamiento de bases de datos paralelas: equilibrio de carga, coherencia de caché y comunicación interprocesador.
- Consultas globales y transacciones.
- Papel del administrador de base de datos local, el administrador de base de datos distribuida y el diccionario global en la arquitectura de componente de un DBMS distribuido.
- Arquitecturas de esquema para DBMS distribuidos firmemente integrados y con integración floja.
- Relación de niveles de transparencia de base de datos distribuida e independencia de datos.
- Tipos de fragmentación: horizontal, vertical y horizontal derivada.
- Complejidad del diseño y asignación de fragmentos.
- Formulación de consultas para transparencia de fragmentación, transparencia de ubicación y transparencia de mapeo local.
- Mediciones y objetivos de rendimiento para procesamiento de consulta distribuida.
- Uso de candado en dos fases para control de concurrencia distribuida.
- Uso del protocolo de copia primaria para reducir gastos generales de actualización con datos copiados.
- Tipos adicionales de fallas en un entorno de base de datos distribuida.
- Protocolo de compromiso en dos fases para asegurar compromiso atómico de sitios participantes en una transacción distribuida.
- Negociaciones entre coordinación centralizada y distribuida para control de concurrencia distribuida y recuperación.

---

## Preguntas

1. ¿Cuál es el papel de los clientes y servidores en el procesamiento distribuido?
2. Defina brevemente los términos flexibilidad, interoperabilidad y escalabilidad. ¿Cómo es que el procesamiento cliente-servidor soporta interoperabilidad, flexibilidad y escalabilidad?
3. Discuta algunos de los equívocos al desarrollar sistemas cliente-servidor.
4. Defina brevemente los términos escalamiento y aceleración y la medición de estos términos.

5. Defina alta disponibilidad e indique cómo el procesamiento de bases de datos paralelas soporta la alta disponibilidad.
6. ¿Cómo puede una base de datos distribuida mejorar el control de datos?
7. ¿Cómo puede una base de datos distribuida reducir los costos de comunicación y mejorar el rendimiento?
8. Discuta algunos de los equívocos cuando se desarrollan bases de datos distribuidas.
9. Discuta por qué el procesamiento distribuido es más maduro y más ampliamente implementado que las bases de datos distribuidas.
10. ¿Por qué son importantes las divisiones de procesamiento y gestión de proceso en las arquitecturas cliente-servidor?
11. Explique cómo las arquitecturas en dos niveles abordan la división del procesamiento y la gestión de proceso.
12. Explique cómo las arquitecturas en tres niveles abordan la división de procesamiento y gestión de proceso.
13. Explique cómo las arquitecturas en niveles múltiples abordan la división de procesamiento y gestión de proceso.
14. ¿Qué es un cliente fino? ¿Cómo se relaciona un cliente fino con la división de procesamiento en arquitecturas cliente-servidor?
15. Mencione algunas razones para elegir una arquitectura en dos niveles.
16. Mencione algunas razones para elegir una arquitectura en tres niveles.
17. Menciona algunas razones para elegir una arquitectura en niveles múltiples.
18. ¿Qué es la arquitectura de servicios web?
19. ¿Cómo soporta interoperabilidad la arquitectura de servicios web?
20. Describa brevemente las arquitecturas básicas para procesamiento de bases de datos paralelas.
21. Describa brevemente las extensiones de clúster a las arquitecturas básicas de bases de datos distribuidas.
22. ¿Cuáles son los principales conflictos de diseño para procesamiento de bases de datos paralelas? Identifique la arquitectura más afectada por los conflictos de diseño.
23. ¿Cuál es el problema de coherencia de caché?
24. ¿Qué es equilibrio de carga?
25. ¿Qué arquitectura de base de datos paralela es soportada por Oracle Real Application Clusters? ¿Cuál es una tecnología clave en Oracle Real Application Clusters?
26. ¿Qué arquitectura de base de datos paralela es soportada por la opción DPF DB2 de IBM? ¿Cuál es la tecnología clave en el DPF?
27. ¿Qué es una solicitud global?
28. ¿Cómo afecta el nivel de integración del DBMS distribuido la arquitectura de componente?
29. ¿Cuándo es apropiado un DBMS distribuido firmemente integrado? ¿Cuándo es apropiado un DBMS distribuido con integración floja?
30. Discuta las diferencias en la arquitectura de esquema para DBMS distribuidos firmemente integrados y con integración floja.
31. ¿Cómo se relaciona la transparencia de base de datos distribuida con la independencia de datos?
32. ¿Siempre es preferible un nivel mayor de transparencia de distribución? Explique brevemente por qué sí o por qué no.
33. ¿Qué es un fragmento horizontal derivado y por qué es útil? ¿Cuál es la relación del operador semi-unión y la fragmentación horizontal derivada?
34. ¿Cuál es la diferencia más grande en la formulación de consulta: (1) transparencia de fragmentación a transparencia de ubicación o (2) transparencia de ubicación a transparencia de mapeo local? Justifique su respuesta.
35. ¿Por qué el diseño y la asignación de fragmento es una tarea compleja?
36. ¿Por qué es importante la optimización de consulta global?
37. ¿Cuáles son las diferencias entre optimización global y local en el procesamiento de consulta distribuida?
38. ¿Por qué hay múltiples objetivos en el procesamiento de consulta distribuida? ¿Cuál objetivo parece ser el más importante?

39. ¿Cuáles son los componentes de las mediciones de rendimiento para procesamiento de consulta distribuida? ¿Qué factores influyen en cómo pueden combinarse estos componentes en una medición de rendimiento?
40. ¿Cómo difiere el candado en dos fases para bases de datos distribuidas con el candado en dos fases para bases de datos centralizadas?
41. ¿Por qué es tan usado el protocolo de copia primaria?
42. ¿Qué tipos de fallas adicionales ocurren en un entorno de base de datos distribuida? ¿Cómo pueden detectarse estas fallas?
43. ¿Cuál es la diferencia entre las fases de votación y de decisión del protocolo de compromiso en dos fases?
44. Discuta las negociaciones entre coordinación centralizada y distribuida en el control de concurrencia y recuperación distribuida.
45. ¿Qué nivel de transparencia proporcionan las bases de datos distribuidas de Oracle?
46. ¿Cuáles son los vínculos de base de datos en el procesamiento de bases de datos distribuidas de Oracle?

## Problemas

Los problemas proporcionan práctica con definición de fragmentos, formulación de consultas en varios niveles de transparencia y definición de estrategias para procesamiento de consulta distribuida. Las preguntas usan las siguientes tablas de base de datos revisadas de una universidad. Esta base de datos es similar a la base de datos usada en el capítulo 4, excepto por las columnas adicionales campus en las tablas *Student*, *Offering* y *Faculty*.

```

Student(StdNo, StdName, StdCampus, StdCity, StdState, StdZip, StdMajor, StdYear)
Course(CourseNo, CrsDesc, CrsCredits)
Offering(OfferNo, CourseNo, OffCampus, OffTerm, OffYear, OffLocation, OffTime, OffDays,
    FacNo)
    FOREIGN KEY CourseNo REFERENCES Course
    FOREIGN KEY FacNo REFERENCES Faculty
Enrollment(OfferNo, StdNo, EnrGrade)
    FOREIGN KEY OfferNo REFERENCES Offering
    FOREIGN KEY StdNo REFERENCES Student
Faculty(FacNo, FacName, FacCampus, FacDept, FacPhone, FacSalary, FacRank)

```

1. Escriba enunciados SQL SELECT para definir dos fragmentos horizontales para estudiantes que asisten 1) al campus Boulder y 2) al campus Denver.
2. Escriba enunciados SQL SELECT para definir dos fragmentos horizontales para profesores que enseñan en 1) el campus Boulder y 2) el campus Denver.
3. Escriba enunciados SQL SELECT para definir dos fragmentos horizontales para cursos ofrecidos en 1) el campus Boulder y 2) el campus Denver.
4. Escriba enunciados SQL SELECT para definir dos fragmentos horizontales derivados para matrículas asociadas con los cursos ofrecidos en 1) el campus Boulder y 2) el campus Denver.
5. Escriba un enunciado SELECT para hacer una lista de los cursos de sistemas de información ofrecidos en el trimestre de primavera 2006. Los cursos de sistemas de información contienen la cadena “IS” en la descripción del curso. Incluya en el resultado el número de curso, descripción, número de oferta y fecha. Suponga transparencia de fragmentación en su formulación.
6. Escriba un enunciado SELECT para hacer una lista de los cursos de sistemas de información ofrecidos en el trimestre de primavera 2006. Los cursos de sistemas de información contienen la cadena “IS” en la descripción del curso. Incluya en el resultado el número de curso, descripción, número de oferta y fecha. Suponga transparencia de ubicación en su formulación.
7. Escriba un enunciado SELECT para hacer una lista de los cursos de sistemas de información ofrecidos en el trimestre de primavera 2006. Los cursos de sistemas de información contienen la cadena “IS” en la descripción del curso. Incluya en el resultado el número de curso, descripción, número de oferta y fecha. Suponga transparencia de mapeo local en su formulación. Los fragmentos *Offering* se asignan a los sitios Boulder y Denver. La tabla *Course* se copia en ambos sitios.
8. Mueva el número de oferta O1 del campus Boulder a Denver. Además del movimiento de oferta entre campus, cambie su ubicación a Plaza 112. Suponga transparencia de fragmentación en su formulación.

**TABLA 17.11**  
Estadísticas para el problema 11

La longitud del registro es 1 000 bits para cada tabla.
32 bits para <i>FacNo</i> .
20 profesores de sistemas de información.
5 000 ofertas para la primavera de 2006.
10 ofertas para la primavera de 2006 en Boulder impartidos por profesores de Denver.
4 000 cursos, 20 000 ofertas y 2 000 profesores en los fragmentos.
La tabla <i>Course</i> se copia en los sitios Denver y Boulder.
El fragmento <i>BoulderOfferings</i> se ubica en el sitio Boulder.
El fragmento <i>DenverFaculty</i> se ubica en el sitio Denver.
El retardo por mensaje es de 0.1 segundo.
La tasa de datos es de 100,000 bits por segundo.

9. Mueva el número de oferta O1 del campus Boulder a Denver. Además del movimiento de oferta entre campus, cambie su ubicación a Plaza 112. Suponga transparencia de ubicación en su formulación.
10. Mueva el número de oferta O1 del campus Boulder a Denver. Además del movimiento de oferta entre campus, cambie su ubicación a Plaza 112. Suponga transparencia de mapeo local en su formulación.
11. Para la siguiente consulta, calcule el tiempo de comunicación (*CT*) para los planes de acceso distribuidos que se mencionan a continuación. Utilice en sus cálculos las fórmulas de la sección 17.6.1 y las estadísticas de consulta y red (véase tabla 17.11).

```
SELECT Course.CourseNo, CrsDesc, OfferNo, OffTime, FacName
  FROM BoulderOfferings BOF, Course, DenverFaculty DF
 WHERE Course.CourseNo = BOF.Course AND OffTerm = 'Spring'
   AND OffYear = 2006 AND DF.FacNo = BF.FacNo
   AND FacDept = 'Information Systems'
```

- 11.1 Mueva todo el fragmento *DenverFaculty* al sitio Boulder y realice la consulta.
- 11.2 Mueva todo el fragmento *BoulderOfferings* al sitio Denver y realice la consulta.
- 11.3 Mueva el fragmento restringido *BoulderOfferings* al sitio Denver y realice la consulta.
- 11.4 Mueva el fragmento restringido *DenverFaculty* al sitio Boulder y realice la consulta.
- 11.5 Restringa el fragmento *DenverFaculty* y mueva los valores de combinación (*FacNo*) al sitio Boulder. Una los valores *FacNo* con la tabla *Course* y el fragmento *BoulderOfferings* en el sitio Boulder. Mueva el resultado de vuelta al sitio Denver para combinar con el fragmento *DenverFaculty*.
12. Investigue las características cliente-servidor, base de datos paralela y base de datos distribuidas de un gran DBMS. Identifique las arquitecturas utilizadas y características importantes del producto.

## Referencias para ampliar su estudio

Aunque este capítulo proporciona una amplia cobertura del procesamiento y datos distribuidos, sólo cubre lo básico. Los libros especializados acerca de la gestión de base de datos distribuidas incluyen el clásico de Ceri y Pelagatti (1984) y el libro más reciente de Ozsu y Valduriez (1991). El libro de Ceri y Pelagatti tiene un capítulo bien escrito acerca del diseño de bases de datos distribuidas. Date (1990) presenta 12 objetivos para sistemas distribuidos con mayor énfasis en DBMSs distribuidos. Bernstein (1996) proporciona una presentación detallada del papel de los middleware en arquitecturas cliente-servidor. El sitio *DBAZine* ([www.dbazine.com](http://www.dbazine.com)) y la *DevX Database Zone* ([www.devx.com](http://www.devx.com)) tienen consejos prácticos acerca del procesamiento distribuido y bases de datos distribuidas.



# Capítulo

# 18

---

# Sistemas de administración de bases de datos de objetos

## Objetivos de aprendizaje

Este capítulo describe las extensiones a los DBMS para soportar operaciones y datos complejos. Después de este capítulo, el estudiante habrá adquirido los siguientes conocimientos y habilidades:

- Citar las razones empresariales para el uso de la tecnología de bases de datos de objetos.
- Definir los principios de la computación orientada a objetos.
- Comparar y contrastar las arquitecturas para la administración de bases de datos de objetos.
- Comprender las características en SQL:2003 para definir y manipular tipos definidos por el usuario, tablas tipo y familias de subtablas.
- Comprender las características en Oracle 10g para definir y manipular tipos definidos por el usuario y tablas tipo.

## Panorama general

---

En el capítulo 17 se describió el procesamiento cliente-servidor para utilizar capacidades de procesamiento remoto y redes de computadoras. Un aumento en la cantidad de procesamiento cliente-servidor implica datos y operaciones complejos que no soportan los DBMS. En muchos casos, el procesamiento cliente-servidor se puede mejorar si, con los datos tradicionales, se integran más cercanamente nuevos tipos de datos y operaciones. En este capítulo aprenderá acerca de las extensiones para DBMS para objetos, combinaciones de datos y operaciones complejas.

Este capítulo proporciona una amplia introducción a los DBMS objeto. Primero aprenderá acerca de las razones empresariales para extender la tecnología de bases de datos. Este capítulo discute el creciente uso de los datos complejos y las incompatibilidades entre los DBMS y los lenguajes de programación como fuerzas impulsoras detrás de la tecnología de bases de datos de objeto. Después de comprender esta motivación, estará listo para aprender acerca de la tecnología de objetos y su impacto sobre los DBMS. Aprenderá sobre los principios de la computación orientada a objetos y las arquitecturas DBMS para soportar estos principios. Este capítulo presenta herencia, encapsulación y polimorfismo como los principios subyacentes de la tecnología

de objetos. Este capítulo presenta cinco arquitecturas para DBMS objeto con el fin de apoyar estos principios. La última parte de este capítulo presenta el soporte a objetos en SQL:2003, el estándar emergente para DBMS orientados a objetos y Oracle 10g, una significativa implementación del estándar SQL:2003. Aprenderá acerca de las características de definición y manipulación de datos para bases de datos de objeto relacional.

## 18.1 Motivación para la administración de bases de datos de objetos

---

Esta sección discute dos fuerzas impulsoras de la demanda por gestión de bases de datos de objetos. Después de una discusión de estas fuerzas, se presentan aplicaciones de ejemplo para mostrar la necesidad de la gestión de bases de datos de objetos.

### 18.1.1 Datos complejos

La mayoría de los DBMS relacionales sólo soportan unos cuantos tipos de datos. Los tipos de datos interconstruidos que soporta SQL incluyen números enteros, números reales, números de precisión fija (monedas), fechas, tiempos y texto. Estos tipos de dato son suficientes para muchas aplicaciones empresariales, o al menos para partes significativas de muchas aplicaciones. Muchas bases de datos empresariales contienen columnas para nombres, precios, direcciones y fechas de transacción que se ajustan fácilmente a los tipos de dato estándar.

Los avances en capacidades de hardware y software han permitido la captura y manipulación de datos complejos en un formato digital. Casi cualquier tipo de dato complejo puede almacenarse digitalmente: imágenes, audio, video, mapas y gráficos tridimensionales. Por ejemplo, una imagen puede representarse como un arreglo bidimensional de pixeles (elementos de imagen) donde cada pixel contiene una propiedad numérica que representa su color o escala de grises. El almacenamiento digital usualmente es más barato y más confiable que los medios tradicionales, como papel, película o diapositivas. Además, el almacenamiento digital facilita la recuperación y manipulación. Por ejemplo, pueden recuperarse imágenes digitales por contenido y similitud con otras imágenes. Las imágenes digitales pueden manipularse en un editor de imágenes con operaciones como recorte, texturizado y ajuste de color.

La capacidad de almacenar y manipular datos complejos no impulsa por sí misma la demanda de tecnología de bases de datos de objetos. Más bien lo que impulsa la demanda de tecnología de bases de datos de objetos es la necesidad de almacenar grandes cantidades de datos complejos y la integración de datos complejos con datos simples. Muchas aplicaciones empresariales requieren grandes cantidades de datos complejos. Por ejemplo, el procesamiento de demandas de seguros y los registros médicos pueden implicar grandes cantidades de datos de imágenes. Almacenar imágenes en archivos separados se vuelve tedioso para una gran colección de imágenes.

La capacidad de usar de manera conjunta datos estándar y complejos es cada vez más importante en muchas aplicaciones empresariales. Por ejemplo, para revisar la condición de un paciente, un médico puede solicitar rayos X junto con estadísticas vitales. Sin la integración, se requieren dos programas separados para mostrar los datos: un editor de imágenes para mostrar los rayos X y un DBMS para recuperar las estadísticas vitales. La capacidad de recuperar tanto imágenes como estadísticas vitales en una sola solicitud es una enorme mejoría. Además de recuperar datos complejos, también puede necesitarse nuevas operaciones. Por ejemplo, un médico puede querer comparar la similitud de los rayos X de un paciente con radiografías que muestren anomalías.

### 18.1.2 Desajuste en el sistema de tipos

El software escrito en lenguaje procedural requiere de acceso a una base de datos cada vez con mayor frecuencia. Los lenguajes procedurales soportan interfaces a la medida para formularios y reportes de datos, operaciones más allá de la capacidad de SQL y procesamiento por lotes. Por ejemplo, con frecuencia es necesario escribir un programa de cómputo para operar sobre datos autorreferenciados como una jerarquía de partes. Usualmente se usa SQL embebido para acceder a una base de datos desde dentro de un programa de cómputo. Después de ejecutar un

enunciado SQL embebido, las columnas de la base de datos se almacenan en variables de programa que pueden manipularse más adelante.

Un desajuste entre los tipos de datos en un DBMS relacional y los tipos de datos de un lenguaje de programación hace al software más complejo y difícil de desarrollar. Por ejemplo, el procesamiento de nómina puede involucrar muchos tipos de intereses, deducciones y ordenamientos de compensación. Una base de datos relacional puede tener una representación para intereses, deducciones y ordenamientos de compensación, mientras que un lenguaje de programación puede tener una representación muy diferente. Antes de codificar los cálculos complejos, los datos deben transformarse de la representación en base de datos relacional (tablas) a la representación del lenguaje de programación (registros u objetos). Después de los cálculos, los datos deben transformarse nuevamente a la representación de base de datos relacional.

El desajuste en los tipos de datos es incluso más pronunciado para los datos complejos. Los lenguajes de programación usualmente tienen sistemas de tipo enriquecido en relación con los DBMS. Por ejemplo, las bases de datos relacionales proporcionan una representación tediosa de formas geométricas en un plano de un edificio. Los objetos como puntos, líneas y polígonos pueden representarse como una columna de texto o como muchas columnas numéricas, como las coordenadas  $X$  y  $Y$  para puntos. En contraste, un lenguaje de programación puede tener tipos de datos a la medida para puntos, líneas y polígonos. Puede haber considerable codificación para transformar entre la representación de base de datos relacional y la representación en lenguaje de programación.

Además, un DBMS relacional no puede realizar operaciones elementales sobre datos complejos. Por ende, en lugar de usar un lenguaje de consulta, debe escribirse un programa de cómputo. Por ejemplo, debe escribirse un programa complejo para calcular la similitud entre dos imágenes. Probablemente el programa contendrá de 10 a 100 veces la cantidad de código que se encuentra en una consulta. Además, el programa debe transformar entre las representaciones en base de datos y lenguaje de programación.

### 18.1.3 Ejemplos de aplicación

Esta sección muestra varias aplicaciones que involucran una mezcla de datos simples y complejos, así como consultas *ad hoc*. Estas aplicaciones tienen características que se encuentran cada vez más en muchas aplicaciones empresariales. Como verá, los DBMSs de objetos soportan los requerimientos de estos tipos de aplicaciones.

#### *Apoyo a consultorio dental*

Los consultorios dentales usan una mezcla de datos simples y complejos para realizar citas, generar facturas y llevar a cabo exámenes. El establecimiento de citas requiere un calendario con bloques de tiempo para proveedores de servicio (dentistas e higienistas). Para la elaboración de exámenes, los proveedores de servicios usan gráficos dentales (gráfico de la boca con la identificación de cada diente), rayos X, hechos del paciente e historias dentales. Después de un examen, la preparación de la factura utiliza la lista de servicios en el examen y los datos de seguro del paciente. Las consultas que implican tanto datos simples como complejos incluyen la muestra de un gráfico dental con los problemas dentales recientes resaltados y la comparación de rayos X para síntomas de daño gingival.

#### *Servicio de listado de bienes raíces*

Los servicios de listado de bienes raíces usan cada vez más datos complejos para facilitar las búsquedas del cliente. Un listado de bienes raíces incluye una mezcla de datos simples y complejos. Los datos simples incluyen numerosos hechos acerca de las casas, como el número de recámaras, los metros cuadrados y el precio de lista. Los datos complejos incluyen fotografías de casas, planos, recorridos en video y mapas del área. Las consultas pueden implicar una mezcla de datos simples y complejos. Los clientes querrán ver casas en un vecindario específico con características seleccionadas. Algunos clientes querrán ver casas con la equiparación más cercana a un conjunto de características ideales donde el cliente asigna un peso a cada característica. Después de seleccionar un conjunto de casas, un cliente querrá explorar la apariencia, plano y hechos acerca de las casas.

### *Reclamaciones por seguro automovilístico*

Las compañías de seguros automovilísticos usan datos complejos para resolver las demandas. El análisis de las demandas implica datos complejos, como fotografías, reportes del accidente y declaraciones de los testigos, así como datos simples, como el conductor y descripciones del vehículo. La resolución de las denuncias implica un mapa que muestre a los proveedores del servicio, así como las tarifas del proveedor de servicio e historia del servicio. Las consultas para datos del accidente y una lista de proveedores de servicios en la proximidad de un cliente implican datos simples y complejos.

## 18.2 Principios orientados a objetos

---

Para proporcionar un fundamento a fin de comprender las arquitecturas y características de los DBMS de objetos, esta sección presenta tres principios fundamentales de computación orientada a objetos. Después de presentar los principios, se discute su impacto sobre los DBMS de objetos y los lenguajes de programación orientados a objetos.

### 18.2.1 Encapsulación

Un objeto es una combinación de datos y procedimientos. A veces, a los datos se les refiere como variables y a los procedimientos como métodos. Cada objeto tiene un identificador único que nunca cambia, a menos que el objeto se destruya. Los identificadores de objeto no son visibles para los usuarios. En contraste, las llaves primarias en las bases de datos relacionales sí son visibles.

Las clases contienen colecciones de objetos. Una definición de clase consiste en una colección de variables y métodos. A veces a las variables se les llama variables de instancia para

#### EJEMPLO 18.1

#### Clase Point parcial

```
CLASS Point {
// VARIABLES:
    ATTRIBUTE Real X; // X coordinate
    ATTRIBUTE Real Y; // Y coordinate
// METHODS:
    Float Distance(IN Point aPoint);
    // Calcula la distancia entre 2 puntos
    Boolean Equals (IN Point aPoint);
    // Determina si dos puntos tienen las mismas coordenadas
};
```

#### EJEMPLO 18.2

#### Clase Bond Parcial

```
CLASS Bond {
// VARIABLES:
    ATTRIBUTE Float IntRate; // Interest Rate
    ATTRIBUTE Date Maturity; // Maturity Date
// METHODS:
    Float Yield();
    // Calcula el rendimiento del bono
};
```

distinguiérlas de las variables que se aplican a toda la clase (llamadas variables de clase). El ejemplo 18.1 muestra la clase *Point* con dos variables (*X* y *Y*, que denotan coordenadas cartesianas), y dos métodos (*Distance* y *Equals*).<sup>1</sup> Cada variable tiene un tipo de dato asociado. Cada método tiene una interfaz y una implementación. La interfaz muestra las entradas y las salidas del método. La implementación muestra la codificación detallada. Por brevedad, en los ejemplos no se muestran las implementaciones. El ejemplo 18.2 muestra un ejemplo más orientado a negocio.

### encapsulación

objetos a los que puede accederse sólo a través de sus interfaces.

Encapsulación significa que a los objetos sólo puede accederse a través de sus interfaces. Los detalles internos (definiciones de variable e implementaciones de método) no son accesibles. Por ejemplo, usted puede usar los métodos *Distance* y *Equals* sólo para manipular los objetos *Point*. Para usar el método *Distance*, proporciona dos objetos *Point*. El primer objeto *Point* está implícito en la interfaz porque *Distance* es un método de *Point*. Para hacer más utilizables los objetos *Point*, debe haber métodos adicionales para crear, borrar y mover puntos.

La encapsulación proporciona dos beneficios para administrar la complejidad del software. Primero, una clase es una unidad más grande de reusabilidad que un procedimiento. En muchos casos, las clases pueden reutilizarse en lugar de sólo hacerlo con los procedimientos individuales. Con el uso de clases más simples pueden definirse clases más complejas. Los ejemplos 18.3 y 18.4 muestran clases que usan las clases *Point* y *Bond* definidas en los ejemplos 18.1 y 18.2. Las nuevas clases (*Rectangle* y *Portfolio*) no necesitan recodificar las variables y métodos de las clases *Point* y *Bond*, respectivamente.

---

### EJEMPLO 18.3

#### Clase *Rectangle* con el uso de la clase *Point*

```
CLASS Rectangle {
// VARIABLES:
    ATTRIBUTE Point UpperLeftCorner; // Upper Left Point
    ATTRIBUTE Point LowerRightCorner; // Lower Right Point
// METHODS:
    Float Area();
    // Calcula el área del rectángulo
    Float Length();
    // Calcula la longitud
    Float Height();
    // Calcula la altura
};
```

---

### EJEMPLO 18.4

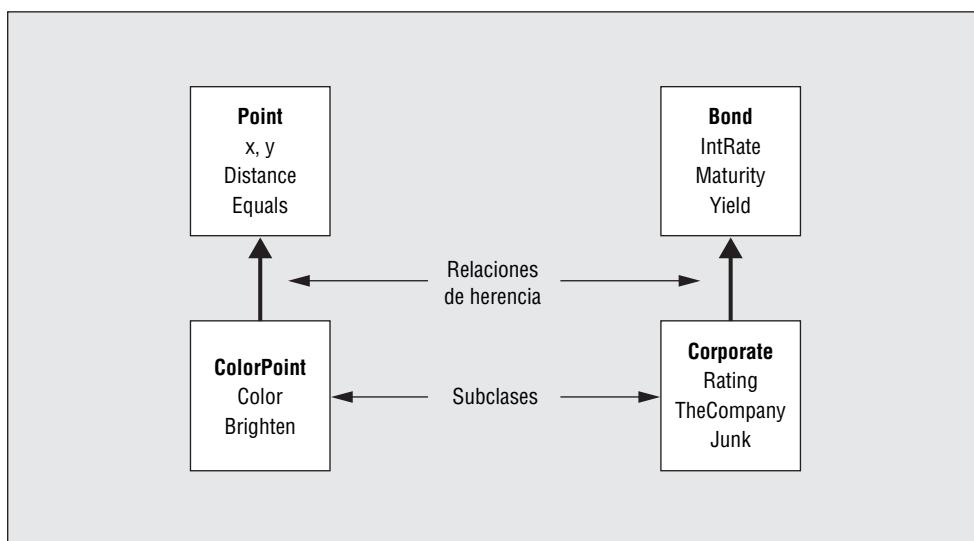
#### Clase *Portfolio* con el uso de la clase *Bond*

```
CLASS Portfolio {
// VARIABLES:
    ATTRIBUTE Set<Bond> BondHoldings; // Set of Bonds
    ATTRIBUTE Set<Stock> StockHoldings; // Set of Stocks
// METHODS:
    Float PortfolioReturn();
    // Calcula el Rendimiento del portafolio
};
```

Como segundo beneficio, la encapsulación proporciona una forma de independencia de datos. Debido a que los detalles internos de los objetos están ocultos, puede hacerse cambios a los detalles internos sin cambiar el código utilizado por los objetos. Recuerde del capítulo 1 que la independencia de datos promueve la reducción en los costos de mantenimiento de software.

<sup>1</sup> Los ejemplos en esta sección se apegan al estándar de Lenguaje de Definición de Objeto (ODL) definido por el Object Database Management Group.

**FIGURA 18.1**  
Jerarquías de clase con las subclases *ColorPoint* y *Corporate*



## 18.2.2 Herencia

### herencia

datos y códigos compartidos entre clases similares (clases y subclases).

El capítulo 5 presentó la clasificación y herencia para el Modelo de Entidad-Relación. Estos conceptos son similares en los modelos orientados a objetos, excepto que la herencia se aplica tanto a datos como a procedimientos. El capítulo 5 describió la herencia sólo para atributos de un tipo entidad. Los ejemplos 18.5 y 18.6 presentan clases que heredan de las clases *Point* y *Bond*. La figura 18.1 muestra una representación gráfica de las relaciones de herencia. Un color point es un punto con color. Del mismo modo, un corporate bond es un bono con una compañía emisora y una calificación de inversión. En las subclases (una clase hijo que hereda de una clase padre), las variables y los métodos de las clases padres no se repiten. Cuando se usan las subclases, pueden usarse los métodos en las clases padre.

### EJEMPLO 18.5

#### Ejemplo *ColorPoint* (subclase de *Point*)

```

CLASS ColorPoint EXTENDS Point {
// VARIABLES:
    ATTRIBUTE Integer Color; // Valor entero que denota un color
// METHODS:
    Integer Brighten(IN Real Intensity);
        // Calcula un nuevo color que sea más brillante
};

```

### EJEMPLO 18.6

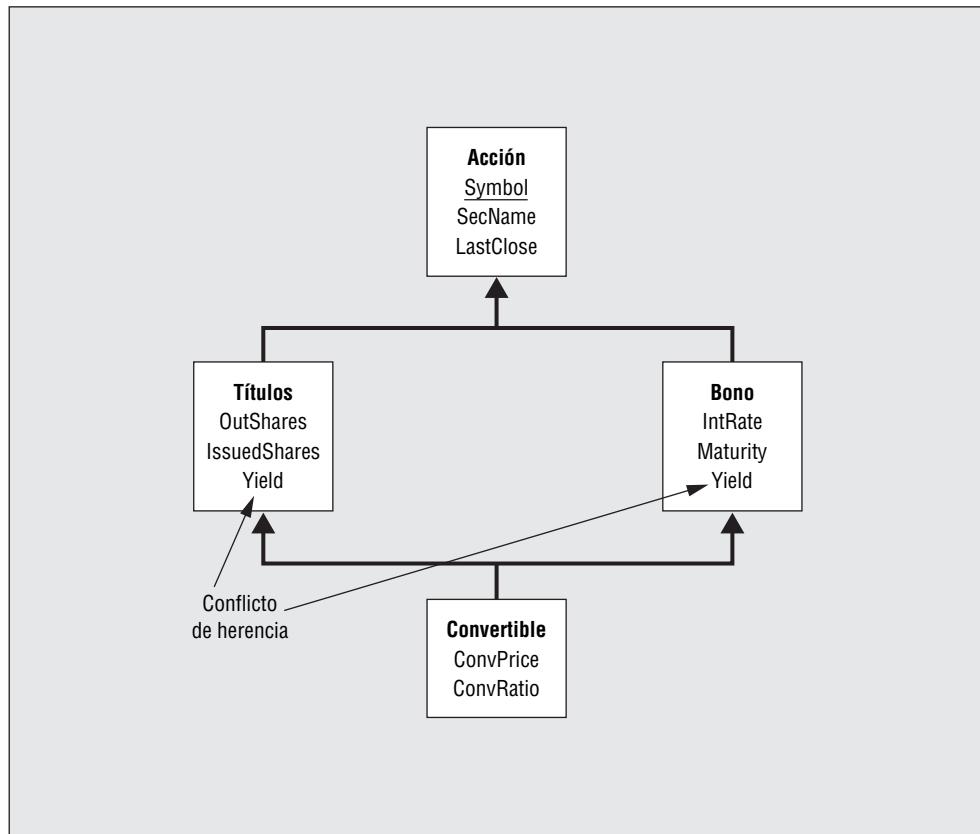
#### Ejemplo de bono *Corporate* (subclase de *Bond*)

```

CLASS Corporate EXTENDS Bond {
// VARIABLES:
    RELATIONSHIP Company TheCompany ; // Compañía emisora del bono
    ATTRIBUTE String Rating; // Calificación de Moody
// METHODS:
    Boolean Junk();
        // TRUE si la calificación del bono significa baja calidad
};

```

**FIGURA 18.2**  
Jerarquía de la clase de inversión



Como se discutió en el capítulo 5, la herencia puede extenderse a múltiples niveles. Por ejemplo, si *Investment* es la clase padre de *Bond*, la clase *Corporate* hereda tanto de *Investment* como de *Bond*. La herencia también puede extenderse a múltiples padres. La figura 18.2 muestra una jerarquía de clase para inversión como herencia múltiple. Un título convertible es un bono que se convierte en acción si lo elige el poseedor. La clase *Convertible* hereda directamente de las clases *Stock* y *Bond*. La herencia múltiple puede ser problemática debido a conflictos. En la figura 18.2, el método *Yield* puede heredarse de *Stock* o de *Bond*. Existen muchos esquemas para resolver conflictos de herencia. A final de cuentas, el diseñador de base de datos debe decidir la clase de la cual hereda.

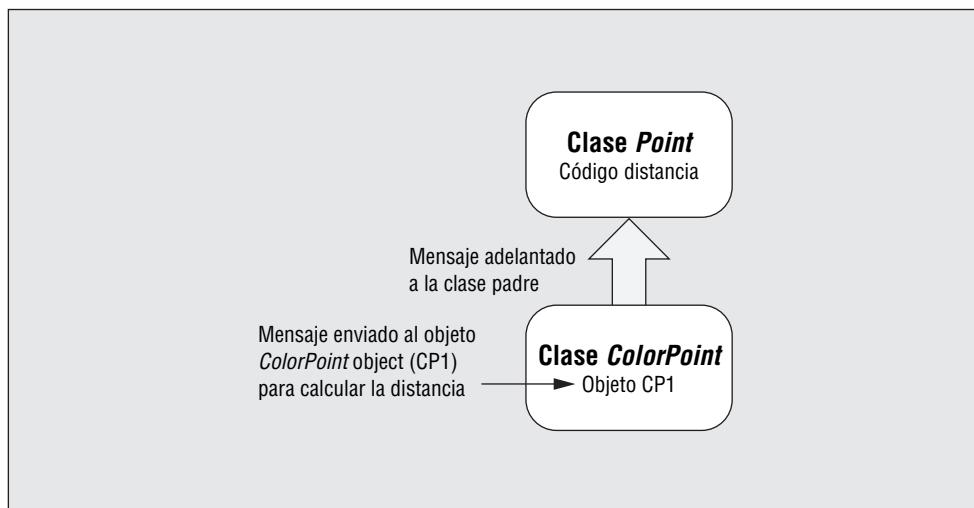
La herencia proporciona una mejor organización de software y de reusabilidad incremental. La organización de clase gráfica muestra similitudes entre clases. Los programadores pueden reducir su tiempo para buscar código similar al usar la organización de clase gráfica. Después de encontrar una clase similar, los diseñadores y programadores de bases de datos pueden agregar incrementalmente más características (variables y métodos). El único código nuevo es sólo para las nuevas características, no para las características existentes.

### 18.2.3 Polimorfismo

**polimorfismo**  
habilidad de un sistema de cómputo para elegir entre múltiples implementaciones de un método.

Polimorfismo literalmente significa “de muchas formas”. En la computación orientada a objetos, el polimorfismo permite un método para tener múltiples implementaciones. La implementación de un método en una subclase puede invalidar la implementación del método en una clase padre. Algunos métodos deben invalidarse porque el significado del método cambia para una subclase. Por ejemplo, el método *Equals* en la clase *ColorPoint* compara las coordenadas y el color mientras que la implementación del método en la clase *Point* sólo compara las coordenadas. En contraste, el método *Area* tiene una implementación más eficiente para la clase *Rectangle* que para la clase padre *Polygon*. En otro ejemplo, el método *Yield* (véase figura 18.2) debe invalidarse en la subclase *Convertible* para combinar los cálculos en las clases padre *Stock* y *Bond*.

**FIGURA 18.3**  
Procesamiento de un  
mensaje



Solicitar la ejecución de un método implica enviar un mensaje a un objeto. Un mensaje contiene un nombre de método y parámetros, similar al llamado de procedimientos. Uno de los parámetros es el objeto que recibe el mensaje. El objeto receptor decide la acción a tomar. Si la clase del objeto contiene el método, se ejecuta. De otro modo, el objeto adelanta el mensaje a su clase padre para su ejecución. En la figura 18.3, el objeto *ColorPoint* recibe un mensaje que solicita un cálculo de distancia. Como la clase *ColorPoint* no contiene una implementación del método *Distance*, el mensaje se envía a la clase padre, *Point*. La clase *Point* ejecuta el método porque contiene una implementación del método *Distance*.

Debido a que la computación orientada a objetos implica mensajes, el procesamiento cliente-servidor y la computación orientada a objetos están íntimamente relacionados. Los clientes y servidores con frecuencia son objetos que se comunican a través de mensajes. El procesamiento cliente-servidor permite que los objetos se ubiquen en diferentes computadoras.

Al procesar un mensaje, el DBMS de objetos supone responsabilidad para elegir la implementación adecuada de un método. Asociar o ligar un mensaje a una implementación de método puede hacerse estáticamente, cuando el código se compila, o dinámicamente, cuando el código se ejecuta. La ligadura estática es más eficiente, pero la ligadura dinámica puede ser más flexible. La ligadura dinámica a veces recibe el nombre de “ligadura final” porque ocurre justo antes de la ejecución del código.

Junto con la ligadura, un DBMS asegura que los objetos y métodos sean compatibles. Esta función se conoce como comprobación de tipo. La comprobación de tipo fuerte asegura que no haya errores de incompatibilidad en el código. Por ejemplo, la comprobación de tipo fuerte evita el cálculo de área para un objeto punto porque el método *Area* se aplica a polígonos, no a puntos. Debido a que las expresiones complejas pueden implicar muchos métodos y objetos, la comprobación de tipo fuerte es un importante tipo de comprobación de consistencia en la computación orientada a objetos.

El polimorfismo soporta menos métodos pero más reutilizables. Como un método puede tener múltiples implementaciones, el número de nombres de método es reducido. Un usuario necesita conocer sólo el nombre del método y la interfaz, no la implementación adecuada que va a utilizar. Por ejemplo, un usuario sólo necesita saber que el método *Area* se aplica a polígonos, no la implementación apropiada para un rectángulo. El DBMS supone responsabilidad para encontrar la implementación adecuada.

El polimorfismo también soporta modificación incremental de código. Al codificar otra implementación de un método para una subclase, con frecuencia puede usarse mucho del código para la implementación del método en la clase padre. Por ejemplo, para codificar la redefinición del método *Equals* en la clase *ColorPoint*, debe agregarse otra condición de igualdad (para el color de un punto) a las condiciones para probar las coordenadas *X* y *Y*.

#### comprobación de tipo fuerte

habilidad para asegurar que el código de programación no contiene errores de incompatibilidad. La comprobación de tipo fuerte es un importante tipo de comprobación de error para la codificación orientada a objetos.

### 18.2.4 Lenguajes de programación versus DBMS

Estos principios se aplican tanto a lenguajes de programación orientados a objetos como a DBMS de objetos. Sin embargo, la aplicación de los principios varía un poco entre lenguajes de programación y DBMS.

Los lenguajes de programación han usado principios orientados a objetos durante muchos años. Simula, un lenguaje desarrollado en los sesenta, es el primer lenguaje reportado de programación orientada a objetos. Como su nombre lo indica, Simula se desarrolló originalmente como un lenguaje de simulación. Los objetos y mensajes son naturales en las simulaciones de modelado. Smalltalk, un lenguaje desarrollado durante los setenta en el Xerox Palo Alto Research Center, fue el primer lenguaje popular de programación orientada a objetos. Smalltalk originalmente ponía énfasis en los objetos para interfaces gráficas de usuario. La herencia entre clases para ventanas y controles es un ajuste natural. A partir del desarrollo de Smalltalk, ha habido muchos otros lenguajes de programación orientados a objetos. Java, C++, Visual Basic y C# son algunos de los lenguajes de programación orientada a objetos más usados en la actualidad.

Los lenguajes de programación orientados a objetos enfatizan el mantenimiento de software y la reutilización del código. La encapsulación se cumple estrictamente para soportar el mantenimiento del software. Los detalles internos (variables e implementaciones del método) no pueden referirse. Además, algunos lenguajes soportan restricciones en los métodos de acceso. Para soportar la reutilización, algunos lenguajes soportan tipos adicionales de herencia para compartir código. La reutilización del código puede extenderse a colecciones de clases, clases con parámetros de tipo dato y código redefinido en una subclase.

Los DBMS de objetos son más recientes que los lenguajes de programación orientados a objetos. La investigación y el desarrollo de DBMS de objetos comenzaron en los ochenta. Hacia principios de los noventa, estaban disponibles comercialmente muchos DBMS de objetos. Además, había un considerable trabajo para extender los DBMS relacionales con nuevas características de objetos. La siguiente sección describe algunas arquitecturas para DBMS de objetos.

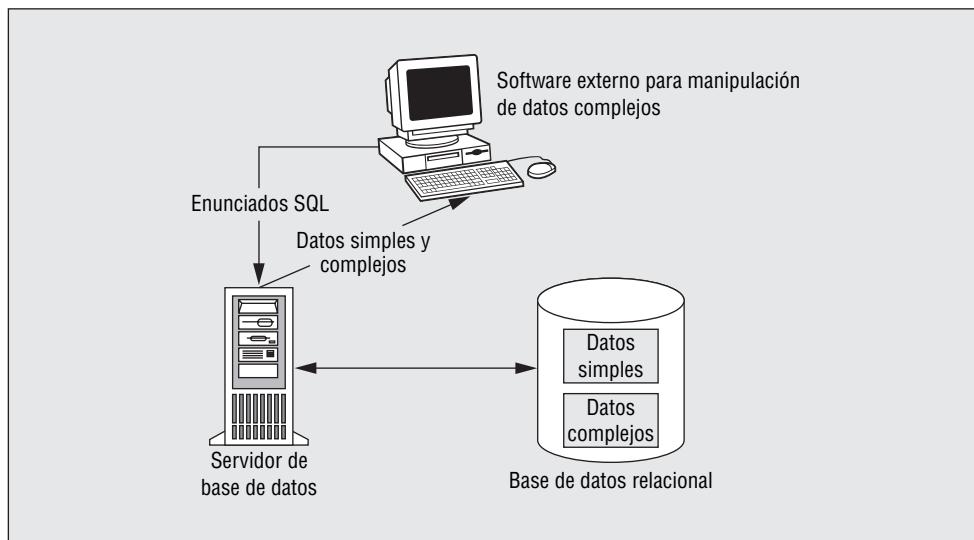
Como los primeros DBMS de objetos comenzaron como extensiones de lenguajes de programación orientados a objetos, no existe el soporte al lenguaje de consultas. En vez de ello, los primeros DBMS de objetos proporcionaron soporte para datos persistentes que duraban más que la ejecución de un programa. Podía accederse a grandes cantidades de datos persistentes, aunque en una forma procedural. Los primeros DBMS de objetos se diseñaron para soportar aplicaciones como el diseño asistido por computadora con grandes cantidades de datos complejos.

La mayoría de los DBMS de objetos soportan ahora acceso a datos no procedurales. Algunos de los principios orientados a objetos son distendidos para facilitar este énfasis. La encapsulación usualmente es distendida, de modo que los datos de un objeto puedan referirse en una consulta. Las características de seguridad de bases de datos proporcionan acceso a características de objeto en lugar de las distinciones público/privado utilizadas en los lenguajes de programación orientada a objetos. Los mecanismos de herencia por lo general son más simples porque se supone que la mayor parte de la codificación es a través de un lenguaje de consulta, no del lenguaje procedural. Generalmente no se soporta la herencia múltiple. Además, la mayoría de los DBMS de objetos ofrecen ahora optimización de consulta y capacidades de procesamiento de transacción.

## 18.3 Arquitecturas para la administración de bases de datos de objetos

El mercado ofrece una diversidad de enfoques para las bases de datos orientadas a objetos. Parte de la variedad es histórica debido al lento desarrollo inicial de la tecnología. Los primeros enfoques proporcionan pequeñas extensiones que dejan la mayor parte del procesamiento orientado a objetos afuera de un DBMS. Conforme crece el interés del usuario y maduran la investigación y el desarrollo, surgen enfoques más completos para las bases de datos orientadas a objetos. Los enfoques recientes han implicado significativas reescrituras de los DBMS para acomodar los

**FIGURA 18.4**  
Arquitectura de objeto grande



objetos. Probablemente el mercado soportará una variedad de enfoques debido a los diferentes requerimientos de los usuarios. Esta sección describe muchas arquitecturas de gestión de bases de datos de objetos, junto con sus fortalezas y debilidades.

#### arquitectura de objeto grande

almacenamiento de objetos grandes (binario o texto) en una base de datos junto con software externo para manipular estos objetos. Los tipos de datos BLOB (objeto grande binario) y CLOB (objeto grande carácter) se usan para almacenar columnas con objetos grandes.

**arquitectura especializada de servidor de medios**  
uso de un servidor dedicado para administrar datos complejos fuera de una base de datos. Los programadores usan una interfaz de programación de aplicación para acceder a datos complejos.

#### 18.3.1 Objetos grandes y software externo

Los primeros enfoques para agregar objetos a las bases de datos relacionales consistieron en usar objetos grandes con software externo. Los datos complejos se almacenan en una columna como objeto grande binario o de texto. Por ejemplo, una imagen usa el tipo de datos BLOB (objeto grande binario), mientras que un gran documento de texto usa el tipo de datos CLOB (objeto grande carácter). Como se muestra en la figura 18.4, los objetos grandes por lo general se almacenan separados de otros datos de la tabla. Una consulta puede recuperar pero no desplegar objetos grandes. El software externo al DBMS despliega y manipula objetos grandes. El software externo incluye controles ActiveX, applets de Java y plug-ins del navegador web.

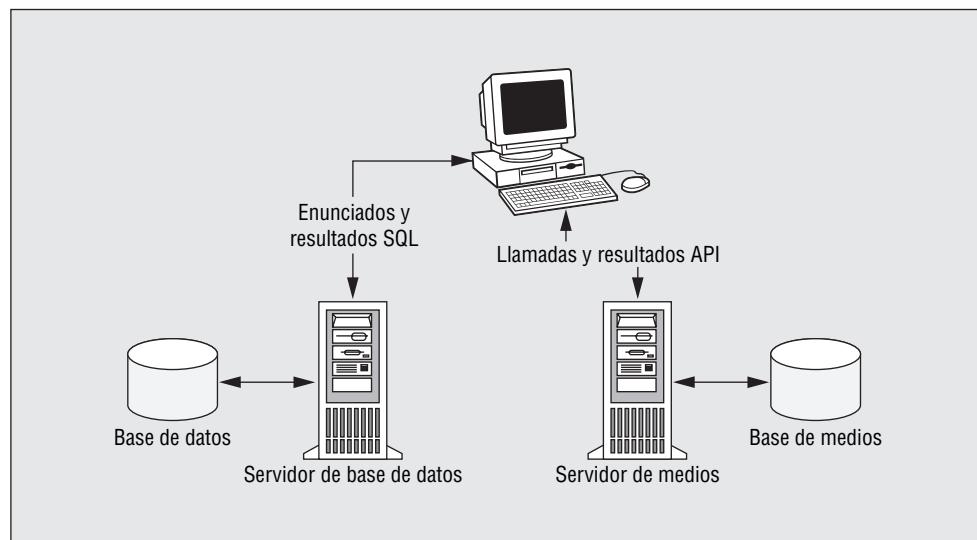
El enfoque de objetos grandes es fácil de implementar y es universal. Sólo se requieren pequeños cambios a un DBMS. Pueden almacenarse todos los tipos de datos complejos. Además, puede haber un gran mercado para software de terceras partes para tipos prominentes de datos complejos. Por ejemplo, se han implementado muchas herramientas de terceros para formatos populares de imágenes.

A pesar de algunas de sus ventajas, el enfoque de objeto grande sufre de serios inconvenientes de rendimiento. Puesto que el DBMS no entiende la estructura y las operaciones de los datos complejos, no es posible la optimización. Los datos no pueden filtrarse con el uso de características de objetos grandes. No pueden usarse índices para seleccionar registros con el uso de características de objetos grandes. Como los objetos grandes se almacenan por separado de otros datos, puede ser necesario accesos adicionales al disco. El orden de los objetos grandes puede no coincidir con el orden de otros datos de la tabla.

#### 18.3.2 Servidores especializados en medios

En la arquitectura especializada de servidor de medios, los datos complejos residen fuera de un DBMS, como se muestra en la figura 18.5. Un servidor separado puede dedicarse a manipular un solo tipo de datos complejos, como video o imágenes. Los programadores usan una interfaz de programación de aplicación (API) para acceder a datos complejos a través de un servidor de medios. La API proporciona un conjunto de procedimientos para recuperar, actualizar y transformar un tipo específico de dato complejo. Para manipular simultáneamente datos simples y complejos, el código de programa contiene una mezcla de SQL y llamadas de API al servidor de medios.

**FIGURA 18.5**  
Arquitectura especializada de servidor de medios



Los servidores especializados de medios proporcionan mejor rendimiento que el enfoque de objeto grande, pero sacrifican cierta flexibilidad. Los servidores dedicados y las API enormemente especializadas proporcionan buen rendimiento para tipos específicos de datos complejos. El rango de operaciones puede ser limitado debido a que en lugar de un lenguaje de consulta se proporciona una API. Por ejemplo, un servidor de video puede soportar transmisión rápida de video, pero no búsqueda por contenido.

Cuando se combinan datos simples y complejos, el acceso al servidor especializado puede tener pobre rendimiento. Un optimizador de consulta no puede optimizar en conjunto la recuperación de datos simples y complejos porque el DBMS no es consciente de los datos complejos. Además, un servidor de medios puede no ofrecer técnicas de indexado para condiciones de búsqueda. El procesamiento de transacción está limitado a datos simples porque los servidores especializados de medios por lo general no soportan procesamiento de transacción.

### 18.3.3 Middleware de bases de datos de objetos

#### middleware de bases de datos de objetos

uso de middleware para gestionar datos complejos posiblemente almacenados fuera de una base de datos junto con los datos tradicionales almacenados en una base de datos.

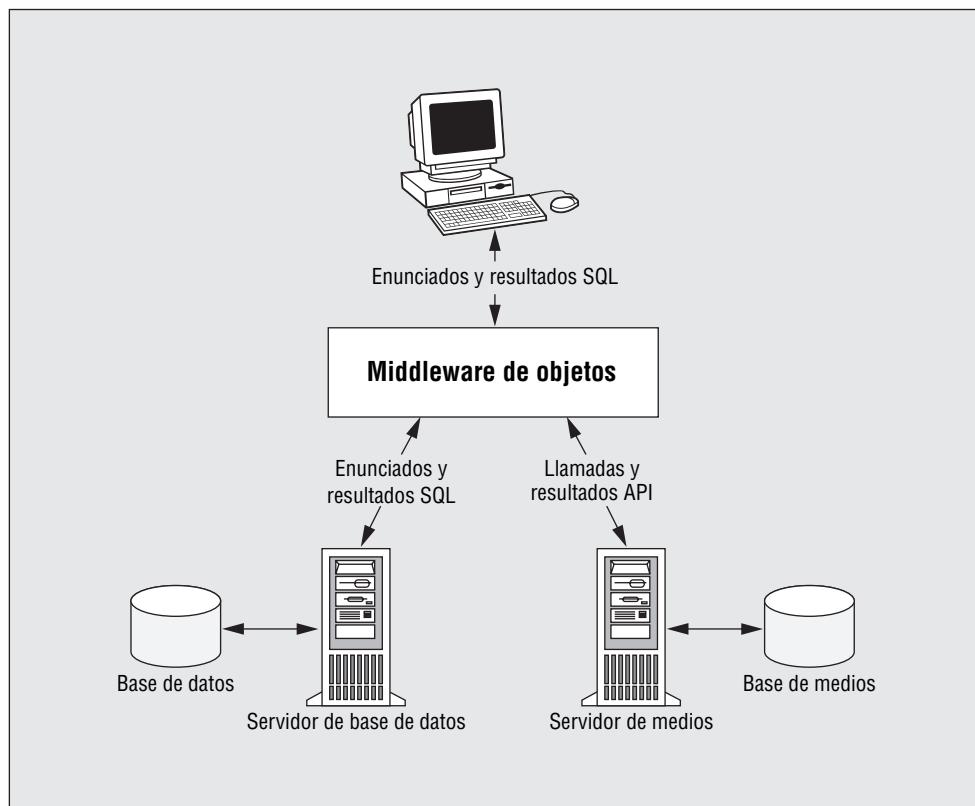
El middleware de base de datos de objetos evade los problemas con los servidores de medios al simular características de objetos. Los clientes ya no acceden directamente a los servidores de medios, como se muestra en la figura 18.6. En vez de ello, los clientes envían enunciados SQL al middleware, quien hace llamadas API a servidores de medios y envía enunciados SQL a los servidores de bases de datos. Los enunciados SQL pueden combinar operaciones tradicionales sobre datos simples con operaciones especializadas sobre datos complejos. El middleware de bases de datos de objetos libera al usuario del conocimiento de una API separada para cada servidor de medios. Además, el middleware de bases de datos de objetos proporciona independencia de ubicación, ya que los usuarios no necesitan saber dónde residen los datos complejos.

El middleware de objetos proporciona una forma para integrar datos complejos almacenados en PC y servidores remotos con bases de datos relacionales. Sin middleware de objeto, algunos de los datos complejos no podrían combinarse fácilmente con datos simples. Incluso si se desea una arquitectura que se integre más firmemente con el DBMS, el enfoque de middleware de objeto puede usarse para datos complejos que los usuarios no deseen almacenar en una base de datos.

El middleware de objetos puede sufrir problemas de rendimiento debido a la falta de integración con el DBMS. La combinación de datos complejos y simples sufre los mismos problemas de rendimiento que los servidores especializados en medios. El DBMS no puede optimizar solicitudes que combinen simultáneamente datos simples y complejos. El middleware puede proporcionar procesamiento de transacción que combine datos simples y complejos; sin embargo, el rendimiento de transacción puede ser lento, pues tiene que usarse compromiso en dos fases y técnicas de control de concurrencia distribuida.

**FIGURA 18.6**

Enfoque de middleware de objetos

**TABLA 18.1**

Tipos de datos pre-construidos definidos por el usuario en DBMS relacionales de objetos

Producto	Tipos definidos por el usuario
IBM DB2 Extenders	Audio, imagen, video, XML, espacial, geodésico, texto, búsqueda en red
Oracle Data Cartridges	Texto, video, imagen, espacial, XML
Informix Data Blades	Texto, imagen, espacial, geodésico, web, series de tiempo, video

### 18.3.4 Sistemas de administración de bases de datos relacionales de objetos para tipos definidos por el usuario

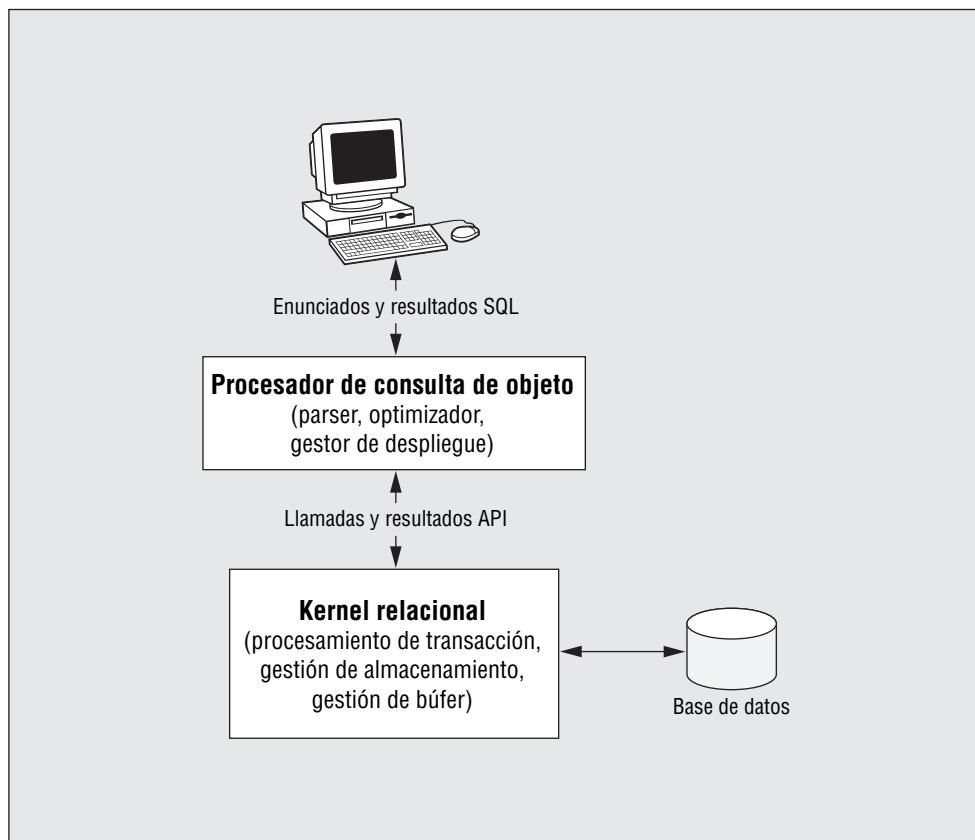
Los primeros tres enfoques implican poco o ningún cambio a un DBMS, pero sólo proporcionan capacidades limitadas de consulta y optimización. Con cambios más grandes al DBMS, es posible tener más capacidades de consulta y optimización.

Para proporcionar capacidades adicionales de consulta, el DBMS relacional de objetos soporta tipos definidos por el usuario. Casi cualquier tipo de dato complejo puede añadirse como un tipo definido por el usuario. Los datos de imagen, los espaciales, las series de tiempo y el video son sólo unos pocos de los posibles tipos de dato. Los grandes proveedores de DBMS ofrecen una colección de tipos preconstruidos definidos por el usuario y la capacidad de extender los tipos preconstruidos, así como crear nuevos tipos definidos por el usuario. La tabla 18.1 menciona tipos de datos preconstruidos soportados por las principales compañías de DBMS. Para cada tipo definido por el usuario, puede definirse una colección de métodos. Estas funciones de método pueden usarse en enunciados SQL, no sólo en código de lenguaje de programación. Herencia y polimorfismo se aplican a tipos de datos definidos por el usuario. Para tipos preconstruidos, se han creado estructuras especializadas de almacenamiento para mejorar el rendimiento. Por ejemplo, pueden proporcionarse Btrees multidimensionales para acceder a datos espaciales.

#### DBMS relacional de objetos

DBMS relacional extendido con un procesador de consulta de objetos para tipos de datos definidos por el usuario. SQL:2003 proporciona el estándar para los DBMS relacionales de objetos.

**FIGURA 18.7**  
Arquitectura de componente para DBMS  
relacionales de objeto

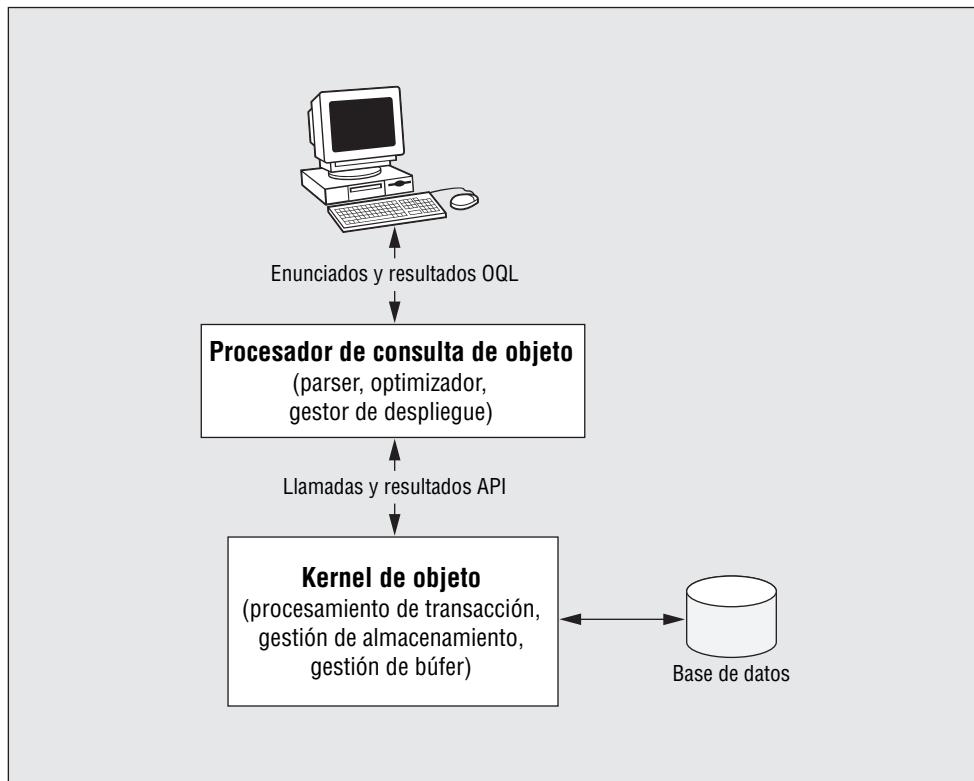


Aunque los tipos definidos por el usuario son las características más destacadas de las bases de datos relacionales de objetos, existen algunas otras características de objeto prominentes que incluyen familias de subtablas (generalización de jerarquías para tablas), matrices y los tipos de datos referencia y fila. La sección 18.4 presenta ejemplos de tipos definidos por el usuario y estas otras características de objeto en SQL:2003, el estándar para DBMS relacionales de objeto. La sección 18.5 presenta las características de objeto en Oracle 10g, una implementación significativa del estándar SQL:2003.

Los tipos definidos por el usuario implican una arquitectura activada por tabla, como se muestra en la figura 18.7. El procesador de consulta de objeto usa código activado por tabla para tipos definidos por el usuario. El parser (análisis gramatical) descompone las referencias a expresiones que involucran tipos y funciones definidas por el usuario. El gestor de despliegue controla la presentación de datos simples y complejos. El optimizador busca estructuras de almacenamiento que puedan usarse para optimizar expresiones que contengan tipos y funciones definidos por el usuario. El kernel relacional comprende procesamiento de transacción, gestión de almacenamiento y gestión de búfer. Proporciona el motor usado por el procesador de consulta de objeto. Para el kernel relacional se necesitan pocos cambios o ninguno.

Los DBMS relacionales de objeto proporcionan buena integración de datos complejos, pero la confiabilidad puede ser una preocupación. La integración de datos simples y complejos implica considerables cambios al parser, al gestor de despliegue y al optimizador. Sin embargo, la base de código en el kernel permanece relativamente estable. La confiabilidad puede comprobarse en las implementaciones de las funciones definidas por el usuario y las estructuras de almacenamiento. Los proveedores de DBMS, los de tercera parte y los desarrolladores locales pueden proporcionar tipos definidos por usuario, que pueden ser complejos y difíciles de implementar. Los errores de implementación pueden afectar la integridad de los datos tanto simples como complejos. Además, los tipos de datos de tercera parte pueden estar expuestos a virus.

**FIGURA 18.8**  
Arquitectura de componente para DBMS orientados a objetos



### 18.3.5 Sistemas de administración de bases de datos orientadas a objetos

Algunos expertos argumentan que es necesario cambios más fundamentales a un DBMS para que soporte objetos. Para acomodar objetos deben cambiarse tanto el modelo de datos como el kernel. Esta convicción impulsó el arranque de algunas compañías de software para implementar una nueva clase de DBMS de objetos, como se muestra en la figura 18.8. Las compañías de software se han aliado para formar Grupos de Administración de Bases de Datos de Objetos (ODMG). Los ODMG proponen un lenguaje de definición de objetos (ODL) y un lenguaje de consulta de objetos (OQL). Estos lenguajes son la contraparte de SQL para los DBMS orientados a objetos.

#### DBMS orientados a objetos

nuevo tipo de DBMS diseñado especialmente para objetos. Los DBMS orientados a objetos tienen un procesador de consulta de objetos y un kernel de objetos. El Grupo de Administración de Datos de Objetos proporciona el estándar para los DBMS orientados a objetos.

Los DBMS orientados a objetos precedieron en 10 años a los ofrecimientos de tecnología relacional de objetos. Los primeros productos se usaron en aplicaciones donde no era importante las consultas, optimización de consulta y procesamiento de transacción *ad hoc*. En vez de ello, los primeros productos enfatizaban el soporte para datos complejos en grandes sistemas de software. La mayoría de los DBMS orientados a objetos comenzaron como lenguajes de programación extendidos con soporte para objetos persistentes (es decir: objetos que existen después de que un programa termina). Gradualmente, los DBMS orientados a objetos proporcionaron consulta, optimización de consulta y eficiente soporte de transacción *ad hoc*. Aun así, todavía quedan dudas acerca de la habilidad de los DBMS orientados a objetos para proporcionar alto rendimiento para aplicaciones empresariales tradicionales y para competir efectivamente en el mercado.

El esfuerzo ODMG se ha eclipsado comercialmente por los estándares relacionales de objeto en SQL:2003. El grupo ODMG se disolvió en 2001 y también ha fallado el desarrollo de DBMS que cumplan el ODMG. Sólo están disponibles unos cuantos productos comerciales y de código abierto. El estándar ODMG fue un desarrollo significativo que fue incapaz de encontrar un nicho comercial significativo. El poder del mercado de los proveedores de DBMS relacionales, el movimiento DBMS de código abierto y el desarrollo de estándares relacionales de objeto suprimieron

**TABLA 18.2 Resumen de arquitecturas**

Arquitectura	Productos ejemplo	Comentarios
Objetos grandes	Mayoría de DBMS SQL-92	Almacenamiento de cualquier tipo de datos; rendimiento incierto; limitado soporte de lenguaje de consulta
Servidores de medios	Oracle interMedia	Limitado soporte de lenguaje de consulta; rendimiento incierto cuando se combinan datos simples y complejos; buen rendimiento en datos complejos
Middleware de bases de datos de objetos	Microsoft Universal Data Access y OLE DB	Habilidad para combinar fuentes de datos distribuidos y diversos; niveles de servicio flexible para fuentes de datos; rendimiento incierto cuando se combinan y actualizan datos simples y complejos
Relacional de objeto (SQL:2003)	IBM Unidata Data Blades, IBM DB2 Extenders, Oracle Data Cartridges; implementaciones significativas de SQL:2003, características relacionales de objeto en Oracle 10g e IBM DB2	Buen rendimiento con estructuras de almacenamiento especializado; buen soporte de lenguaje de consulta; algún tipo de incompatibilidad con lenguajes de programación; razonable conformidad con características de objetos SQL:2003
Orientada a objetos (ODMG)	Object Store, UniSQL, Versant	Buen soporte de lenguaje de consulta; rendimiento incierto para aplicaciones tradicionales; buen tipo de compatibilidad con lenguajes de programación; pocas ofertas comerciales o de código abierto

el desarrollo adicional de los DBMS que cumplen el ODMG. El beneficio agregado de la tecnología de bases de datos orientadas a objetos sobre la tecnología relacional de objetos no ha sido lo suficientemente persuasiva como para generar éxito comercial o de código abierto.

### 18.3.6 Resumen de arquitecturas de bases de datos de objetos

La tabla 18.2 ofrece un resumen práctico para ayudarlo a recordar las arquitecturas para DBMSs de objetos. Todas las arquitecturas satisfacen cierto nicho de mercado. Las arquitecturas más simples (objetos grandes y servidores de medios) se volverán menos populares con el tiempo. La lucha por dominar entre las otras tres arquitecturas puede no decidirse durante algún tiempo. La arquitectura de middleware de bases de datos de objetos probablemente coexistirá con las otras arquitecturas para manejar datos complejos almacenados fuera de una base de datos.

## 18.4 Características de bases de datos de objetos en SQL:2003

Para aclarar los conceptos de bases de datos de objetos, esta sección proporciona ejemplos con el uso de SQL:2003, el lenguaje de bases de datos relacionales de objetos. El estándar SQL:2003 incluye nueve partes y siete paquetes, como se resume en la tabla 18.3. Core SQL:2003 consta de las partes 1, 2 y 11. Cada parte sin núcleo contiene características obligatorias y opcionales. Un paquete es una colección de características opcionales para alguna área de aplicación o entorno de implementación.

El estándar SQL:2003 proporciona varios niveles de conformidad. La conformidad mínima incluye todas las características en Core SQL:2003 (partes 1, 2 y 11). La conformidad mejorada puede afirmarse para las características obligatorias en una parte, las características opcionales en una parte y las características en un paquete. Debido a la falta de suites de prueba y certificación independiente, el nivel de conformidad de un DBMS específico es difícil de determinar. La mayoría de los proveedores proporcionan listas de comprobación de características soportadas, características no soportadas y características soportadas con extensiones propietarias.

Esta sección presenta ejemplos de los paquetes Basic Objeto Support y Enhanced Objeto Support. Los ejemplos demuestran las características de SQL:2003 para tipos de datos definidos por el usuario, definiciones de tablas con tablas tipo, familias de subtablas y uso de tablas tipo.

**TABLA 18.3 Panorama de las partes y paquetes de SQL:2003**

Componente SQL:2003	Ámbito
Parte 1: Marco conceptual	Notación de sintaxis, terminología, resultados de procesamiento de enunciado, niveles de conformidad (Core SQL:2003)
Parte 2: Cimientos	Estructuras de datos y operaciones básicas en datos SQL (Core SQL:2003)
Parte 3: Interfaz de nivel llamada	Estructuras de datos y funciones para el uso de SQL en un programa de cómputo
Parte 4: Módulos almacenados persistentes	Enunciados en lenguaje de programación, procedimientos, funciones, manipulación de excepción
Parte 9: Administración de datos externos	Envolturas de datos extraños y vínculos de datos para datos externos
Parte 10: Ligaduras de lenguaje de objeto	Enunciados a SQL embebido en programas Java
Parte 11: Esquemas de información y definición	Tablas de diccionario para tablas, integridad y seguridad (Core SQL:2003)
Parte 13: Rutinas y tipos SQL con el uso de lenguaje de programación Java TM	Uso de clases y métodos Java en enunciados SQL
Parte 14: Especificaciones relacionadas con XML	Manipulación de documentos XML en enunciados SQL
Paquete 1: Facilidades de fecha mejoradas	Especificación de zona horaria, tipo de datos de intervalo
Paquete 2: Gestión de integridad mejorada	Verificación de restricciones, afirmaciones y administración de la restricción
Paquete 4: Módulos almacenados persistentes	Enunciados de lenguaje de programación, procedimientos, funciones, manipulación de excepción
Paquete 6: Soporte de objeto básico	Tipos de datos definidos por el usuario, herencia sencilla, tipos de referencia, matrices
Paquete 7: Soporte de objeto mejorado	Expresiones de trayectorias, definición de subtabla, búsqueda en subtabla, subtipos
Paquete 8: Bases de datos activas	Soporte para disparadores
Paquete 10: Facilidades OLAP	Operadores de cubo y de roll-up, constructores de fila y tabla, operadores FULL JOIN e INTERSECT

#### 18.4.1 Tipos definidos por el usuario

Una de las extensiones fundamentales en SQL:2003 es el tipo definido por el usuario para atar datos y procedimientos. Los tipos definidos por el usuario soportan la definición de nuevos tipos de datos estructurados, así como el refinamiento de los tipos de datos estándar. Un tipo de datos estructurado tiene una colección de atributos y métodos. En SQL-92, el enunciado CREATE DOMAIN soporta refinamientos a tipos de datos estándar, pero no la definición de nuevos tipos estructurados.

El ejemplo 18.7 muestra el tipo *Point* para contrastar con la notación ODMG que se muestra en el ejemplo 18.1. Algunas diferencias son aparentes, como las palabras clave (TYPE frente a CLASS) y el orden de especificación (el nombre de campo antes del tipo de dato). La primera parte de un tipo definido por el usuario contiene las definiciones de atributos. La doble raya denota un comentario. Para métodos, el primer parámetro está implícito, como la notación ODMG. Por ejemplo, el método *Distance* sólo menciona un parámetro *Point* porque el otro parámetro *Point* está implícito. En SQL:2003, los métodos sólo usan parámetros de entrada y

**EJEMPLO 18.7**
**Tipo *Point* en SQL:2003**

```

CREATE TYPE Point AS
( X FLOAT, -- coordenada X
  Y FLOAT ) -- coordenada Y
METHOD Distance(P2 Point) RETURNS FLOAT,
  -- Calcula la distancia entre 2 puntos
METHOD Equals (P2 Point) RETURNS BOOLEAN
  -- Determina si 2 puntos son equivalentes
NOT FINAL
INSTANTIABLE;
  
```

deben regresar valores. El cuerpo de los métodos no se muestra en el enunciado CREATE TYPE sino más bien en el enunciado CREATE METHOD. Las palabras clave NOT FINAL significan que pueden definirse subtipos. La palabra clave INSTANTIABLE significa que pueden crearse instancias del tipo.<sup>2</sup>

Como se mencionó en el párrafo anterior, los métodos SQL:2003 son un poco limitados en cuanto a que deben regresar valores sencillos y sólo usar parámetros de entrada. Además, el primer parámetro de un método es implícitamente una instancia del tipo en el que está asociado. SQL:2003 proporciona funciones y procedimientos que no tienen las restricciones de los métodos. Como las funciones y procedimientos no están asociados con un tipo específico, SQL:2003 proporciona enunciados de definición separados (CREATE FUNCTION y CREATE PROCEDURE). Los procedimientos pueden usar parámetros de entrada, salida y entrada-salida, mientras que las funciones sólo usan parámetros de entrada.

El ejemplo 18.8 muestra el tipo *ColorPoint*, un subtipo de *Point*. La palabra clave UNDER indica el tipo padre. Debido a que SQL:2003 no soporta herencia múltiples, a la palabra clave UNDER sólo puede seguirle un nombre tipo sencillo. En las definiciones de método, la palabra clave OVERRIDING indica que el método invalida la definición en un tipo padre.

### EJEMPLO 18.8

#### Tipo *ColorPoint*

```
CREATE TYPE ColorPoint UNDER Point AS
(Color INTEGER)
METHOD Brighten (Intensity INTEGER) RETURNS INTEGER,
-- Aumenta intensidad de color
OVERRIDING METHOD Equals (CP2 ColorPoint)
RETURNS BOOLEAN
-- Determina si 2 ColorPoints son equivalentes
FINAL;
```

Además de los métodos explícitos mencionados en el enunciado CREATE TYPE, los tipos definidos por el usuario tienen métodos implícitos que pueden usarse en enunciados SQL y procedimientos almacenados, como se muestra a continuación:

- **Método constructor:** crea una instancia vacía del tipo. El método constructor tiene el mismo nombre que el tipo de datos. Por ejemplo, *Point()* es el método constructor para el tipo *Point*.
- **Métodos de observador:** recuperan valores de los atributos. Cada método observador usa el mismo nombre que su atributo asociado. Por ejemplo, *X()* es el método observador para el atributo *X* del tipo *Point*.
- **Métodos de mutador:** cambian valores almacenados en los atributos. Cada método mutador usa el mismo nombre que su atributo asociado con un parámetro para el valor. Por ejemplo, *X(45.0)* cambia el valor del atributo *X*.

SQL:2003 presenta los tipos de colección ARRAY y MULTISET para soportar tipos con más de un valor, como series de tiempo y formas geométricas. Arreglos soportan colecciones ordenadas acotadas, mientras que multiset soporta colecciones no ordenadas ni acotadas. El ejemplo 18.9a) define un tipo triángulo con un arreglo de tres puntos. El número que sigue a la palabra clave ARRAY indica el tamaño máximo del arreglo. El ejemplo 18.9b) define un polígono con un multiset de puntos. Debe observar que no puede especificarse longitud máxima para atributos MULTISET.

<sup>2</sup> Las palabras clave NOT INSTANTIABLE significan que el tipo es abstracto sin instancias. Los tipos abstractos contienen datos y código pero no instancias. Se ha encontrado que los tipos abstractos mejoran la compartición de código en programación orientada a objetos.

**EJEMPLO 18.9a) Tipo *Triangle* con el uso del tipo ARRAY**

```

CREATE TYPE Triangle AS
  (Corners Point ARRAY[3], -- Arreglo de puntas de esquina
   Color INTEGER )
  METHOD Area() RETURNS FLOAT,
    -- Calcula el área
  METHOD Scale (Factor FLOAT) RETURNS Triangle
    -- Calcula un nuevo triángulo aumentado por factor
  NOT FINAL;

```

**EJEMPLO 18.9b) Tipo *Polygon* con el uso del tipo MULTISET**

```

CREATE TYPE Polygon AS
  (Corners Point MULTISET, -- Multiset de puntas de esquina
   Color INTEGER )
  METHOD Area() RETURNS FLOAT,
    -- Calcula el área
  METHOD Scale (Factor FLOAT) RETURNS Polygon
    -- Calcula un nuevo polígono aumentado por factor
  NOT FINAL;

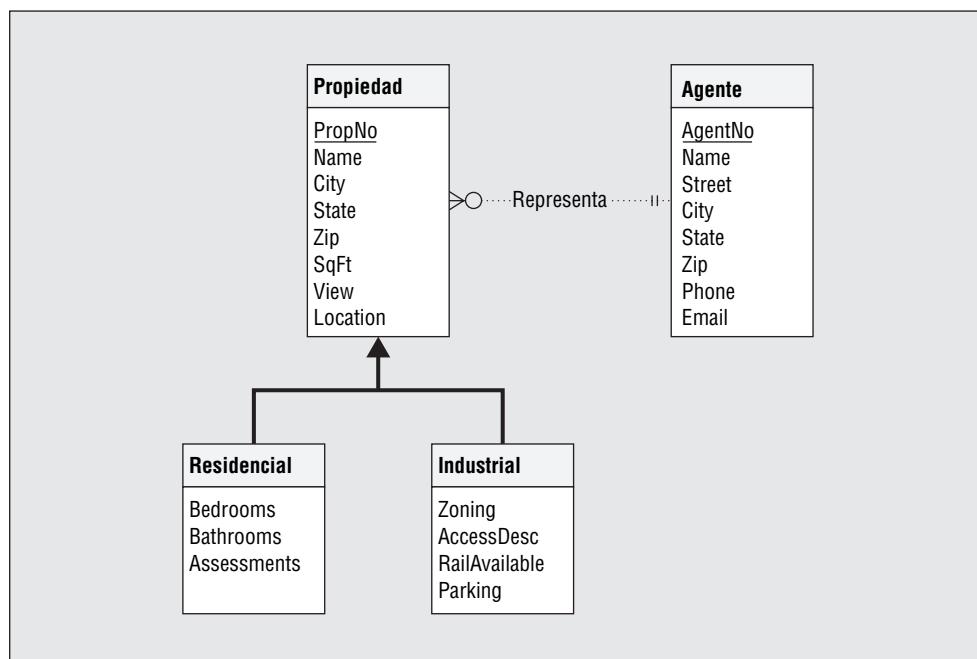
```

Los tipos definidos por el usuario se integran en el corazón de SQL:2003. Los tipos definidos por el usuario pueden usarse como tipos de datos para columnas en tablas, pasar como parámetros y regresar como valores. Los métodos definidos por el usuario pueden usarse en expresiones en las cláusulas SELECT, WHERE y HAVING.

#### 18.4.2 Definiciones de tabla

Los ejemplos en el resto del capítulo 18 se basan en una simple base de datos de propiedad, con propiedades y agentes, como se muestra en el ERD de la figura 18.9. Para la presentación que

**FIGURA 18.9**  
ERD para la base de datos propiedad



aquí se hace, el aspecto más importante del ERD es la jerarquía de generalización para propiedades. SQL:2003 ofrece soporte directo de jerarquías de generalización en lugar de soporte indirecto, como se indica mediante la regla de conversión de jerarquía de generalización presentada en el capítulo 6.

SQL:2003 soporta dos estilos de definiciones de tabla. El estilo SQL-92 tradicional usa llaves foráneas para vincular dos tablas. El ejemplo 18.10 muestra la tabla *Property* con el uso de una llave foránea para hacer referencia al agente que representa la propiedad. La columna *vista* usa el tipo objeto binario grande (BLOB). Proporcionar un tipo preconstruido definido por el usuario para un formato de imagen específico sería una mejor opción que el tipo BLOB.

#### **EJEMPLO 18.10 Tablas *Agent* y *Property* con el uso de estilo de definición SQL-92 tradicional**

```
CREATE TABLE Agent
(AgentNo    INTEGER,
Name        VARCHAR(30),
Street      VARCHAR(50),
City        VARCHAR(30),
State       CHAR(2),
Zip         CHAR(9),
Phone       CHAR(13),
Email       VARCHAR(50),
CONSTRAINT AgentPK PRIMARY KEY(AgentNo)  );

CREATE TABLE Property
(PropNo     INTEGER,
Street      VARCHAR(50),
City        VARCHAR(30),
State       CHAR(2),
Zip         CHAR(9),
SqFt        INTEGER,
View        BLOB,
AgentNo    INTEGER,
Location Point,
CONSTRAINT PropertyPK PRIMARY KEY(PropNo),
CONSTRAINT AgentFK FOREIGN KEY(AgentNo) REFERENCES Agent  );
```

SQL:2003 soporta el constructor de tipo fila para permitir que las filas de una tabla se almacenen como variables, se usen como parámetros y regresen mediante funciones. Un tipo fila es una secuencia de pares nombre/valor. El uso de un tipo fila es juntar columnas relacionadas de modo que puedan almacenarse como una variable o pasar como un parámetro. El ejemplo 18.11 muestra la tabla *Property* con el uso de un tipo fila para las columnas de dirección (*Street*, *City*, *State* y *Zip*).

#### **EJEMPLO 18.11 Revisión de la definición de tabla *Property* con un tipo fila**

```
CREATE TABLE Property
(PropNo    INTEGER,
Address   ROW (Street  VARCHAR(50),
City      VARCHAR(30),
State     CHAR(2),
Zip       CHAR(9) ),
```

```

SqFt      INTEGER,
View       BLOB,
AgentNo   INTEGER,
Location  Point,
CONSTRAINT PropertyPK PRIMARY KEY(PropNo),
CONSTRAINT AgentFK FOREIGN KEY(AgentNo) REFERENCES Agent  );

```

SQL:2003 proporciona un estilo alternativo de definición de tabla conocido como tablas tipo para soportar identificadores de objeto y referencias de objeto. Con las tablas tipo, una definición de tabla hace referencia a un tipo definido por el usuario en vez de proporcionar su propia lista de columnas. El ejemplo 18.12 muestra el tipo definido por el usuario *AgentType* y la tabla *Agent* que se refiere a *AgentType*. Además, se usa *AddressType* (un tipo estructurado de nombre) en lugar del tipo ROW no mencionado en el ejemplo 18.11. La cláusula REF define

**EJEMPLO 18.12 Definición de *AddressType* y *AgentType* seguidos por la tabla tipo *Agent* con base en *AgentType***

```

CREATE TYPE AddressType AS
(Street VARCHAR(50),
City   VARCHAR(30),
State  CHAR(2),
Zip    CHAR(9)  )
NOT FINAL;

CREATE TYPE AgentType AS
(AgentNo INTEGER,
Name   VARCHAR(30),
Address AddressType,
Phone  CHAR(13),
Email  VARCHAR(50)  )
NOT FINAL;

CREATE TABLE Agent OF AgentType
(REF IS AgentOld SYSTEM GENERATED,
CONSTRAINT AgentPK PRIMARY KEY(AgentNo)  );

```

un identificador de objeto para la tabla. Las palabras clave SYSTEM GENERATED indican que los identificadores de objeto son generados por el DBMS, no por la aplicación del usuario (palabras clave USER GENERATED).

Otras tablas pueden hacer referencia a tablas con base en tipos definidos por el usuario. Las referencias a objetos proporcionan una alternativa a referencias de valor para llaves foráneas. El ejemplo 18.13 muestra una definición del tipo *.PropertyType* con una referencia a *AgentType*. La cláusula SCOPE limita una referencia a las filas de una tabla en lugar de a objetos del tipo.

Como muestran estos ejemplos, SQL:2003 proporciona una diversidad de formas para definir tablas (tipo frente a tablas no tipo, referencias frente a llaves foráneas tradicionales, tipos ROW frente a columnas frente a tipos estructurados nominados). Para entrenamiento de los programadores de aplicaciones, es importante el uso consistente de estilos de definición de tabla. Una regla empírica razonable es usar las definiciones de tabla tradicionales (tablas no tipo con columnas no estructuradas y llaves foráneas) o tablas tipo con tipos estructurados nominados y

**EJEMPLO 18.13 Definición de *.PropertyType* y la tabla tipo *Property***

```

CREATE TYPE PropertyType AS
(PropNo INTEGER,
Address AddressType,
SqFt INTEGER,
View BLOB,
Location Point,
AgentRef REF(AgentType) SCOPE Agent )
NOT FINAL;

CREATE TABLE Property OF PropertyType
(REF IS PropertyOld SYSTEM GENERATED,
CONSTRAINT PropertyPK PRIMARY KEY(PropNo) );

```

tipos de referencia. Los estilos de definición de tabla de mezcla pueden agobiar a los programadores de aplicaciones porque el estilo de definición influye la codificación usada en los enunciados de recuperación y manipulación.

SQL:2003 soporta tablas anidadas con el uso del tipo MULTISET con el tipo ROW para elementos de un multiset. Las tablas anidadas son útiles al nivel de aplicación, en especial para reglas empresariales complejas que implican procedimientos almacenados. Además, las tablas anidadas pueden ser útiles para reducir la incompatibilidad de sistema de tipo entre un DBMS y un lenguaje de programación, como se discutió en la sección 18.1.2. En el nivel de diseño de tabla, el uso de tablas anidadas no es claro para bases de datos empresariales. Aunque existe alguna teoría de diseño para las tablas anidadas, la teoría no es ampliamente conocida o practicada. Dada la inmadurez de la práctica de las tablas anidadas, no se presentan ejemplos de tablas anidadas para el diseño de tablas.

**18.4.3 Familias de subtablas**

La herencia se aplica a las tablas de forma similar a como se aplica a los tipos definidos por el usuario. Una tabla puede declararse como subtabla de otra tabla. Una subtabla hereda las columnas de sus tablas padres. SQL:2003 limita la herencia para tablas a herencia sencilla. Una parte potencialmente confusa de la herencia de tabla implica la herencia de tipo. Las tablas involucradas en relaciones de subtabla deben ser tablas tipo con los tipos asociados también participando en las relaciones de subtipo, como se demostró en el ejemplo 18.14. Note que las cláusulas REF y las restricciones de llave primaria son heredadas de la tabla *Property*, de modo que no se especifican para las tablas *Residential* e *Industrial*.

La inclusión de conjunto determina la relación de filas de una tabla padre a las filas de sus subtablas. Cada fila en una subtabla también es una fila en cada una de sus tablas ancestros (padres directos y padres indirectos). Cada fila de una tabla padre corresponde a, cuando mucho,

**EJEMPLO 18.14 Subtipos y subtablas para propiedades residential e industrial**

```

CREATE TYPE ResidentialType UNDER PropertyType
(BedRooms INTEGER,
BathRooms INTEGER,
Assessments DECIMAL(9,2) ARRAY[6] )
NOT FINAL
INSTANTIABLE;

CREATE TABLE Residential OF ResidentialType UNDER Property;

```

```

CREATE TYPE IndustrialType UNDER PropertyType
(Zoning VARCHAR(20),
AccessDesc VARCHAR(20),
RailAvailable BOOLEAN,
Parking VARCHAR(10)  )
NOT FINAL
INSTANTIABLE;
CREATE TABLE Industrial OF IndustrialType UNDER Property;

```

una fila en subtablas directas. Esta relación de inclusión de conjunto se extiende a toda una familia de subtablas, incluidas la tabla raíz y todas las subtablas directa o indirectamente bajo la tabla raíz. Por ejemplo, una familia de subtablas incluye seguridad como la raíz; bono y acción, bajo inversión; y corporativo, municipal y federal bajo bono. La raíz de una familia de subtablas se conoce como tabla máxima. En este ejemplo seguridad es la tabla máxima.

Las operaciones de manipulación de datos en un fila en una familia de subtablas afecta las filas relacionadas en tablas padre y subtablas. Lo siguiente es una breve descripción de los efectos colaterales cuando se manipulan filas en familias de subtablas.

- Si se inserta una fila en una subtabla, entonces se inserta en cada tabla padre una fila correspondiente (con los mismos valores para columnas heredadas). Las inserciones pasan en cascada hacia arriba en la familia de subtablas hasta que llegan a la tabla máxima.
- Si una columna se actualiza en una tabla padre, entonces la columna también se actualiza en todas las subtablas directas e indirectas que heredan la columna.
- Si una columna heredada se actualiza en una subtabla, entonces la columna cambia en las filas correspondientes de las tablas padre directas e indirectas. La actualización en cascada se detiene en la tabla padre en la que la columna se define (no heredada).
- Si se borra una fila en una familia de subtablas, todas las filas correspondientes en las tablas padre y subtablas también se borran.

#### 18.4.4 Manipulación de objetos complejos y familias de subtablas

Las más ricas capacidades de definición de datos de SQL:2003 conducen a nuevas características cuando en la manipulación de datos y en los enunciados de recuperación de datos se usan columnas tipo fila y columnas tipo referencia. Cuando se insertan datos en una tabla con una columna tipo fila, debe usarse la palabra clave ROW, como se muestra en el ejemplo 18.15. Si una columna usa un tipo definido por el usuario en lugar del tipo ROW, debe usarse el nombre tipo, como se muestra en el ejemplo 18.16.

Cuando se insertan datos en una tabla con una columna tipo referencia, el identificador de objeto puede obtenerse con un enunciado SELECT. Si los identificadores de objeto para una tabla referenciada son generados por el usuario (como los valores de llaves primarias), puede no

#### EJEMPLO 18.15 Uso de la palabra clave ROW cuando se inserta una fila en una tabla con una columna tipo fila

Este ejemplo supone que la columna *Address* del tipo *AgentType* se definió con el tipo ROW.

```

INSERT INTO Agent
(AgentNo, Name, Address, Email, Phone)
VALUES (999999, 'Sue Smith',
ROW('123 Any Street', 'Denver', 'CO', '80217'),
'sue.smith@anyisp.com', '13031234567')

```

**EJEMPLO 18.16 Uso del nombre de tipo cuando se insertan dos filas en una tabla con columna tipo estructurado**

Este ejemplo corresponde al tipo *AgentType* definido en el ejemplo 18.12.

```
INSERT INTO Agent
(AgentNo, Name, Address, Email, Phone)
VALUES (999999, 'Sue Smith',
AddressType('123 Any Street', 'Denver', 'CO', '80217'),
'sue.smith@anyisp.com', '13031234567');

INSERT INTO Agent
(AgentNo, Name, Address, Email, Phone)
VALUES (999998, 'John Smith',
AddressType('123 Big Street', 'Boulder', 'CO', '80217'),
'john.smith@anyisp.com', '13034567123');
```

necesitarse un enunciado SELECT. Incluso con identificadores de objeto generados por usuario, puede necesitarse un enunciado SELECT si el identificador de objeto no se conoce cuando se inserta la fila. El ejemplo 18.17 demuestra un enunciado SELECT para recuperar el identificador de objeto (*AgentOID*) de la fila relacionada de la tabla *Agent*. En el enunciado SELECT, los otros valores que se deben insertar son valores constantes. Para la columna *Assessments*, el valor constante es un arreglo de valores denotados por la palabra clave ARRAY junto con los corchetes que rodean los valores de elemento del arreglo.

**EJEMPLO 18.17 Uso de un enunciado SELECT para recuperar el identificador de objeto de la fila Agent relacionada**

```
INSERT INTO Residential
(PropNo, Address, SqFt, AgentRef, BedRooms, BathRooms, Assessments)
SELECT 999999, AddressType('123 Any Street', 'Denver', 'CO', '80217'),
2000, AgentOID, 3, 2, ARRAY[190000, 200000]
FROM Agent
WHERE AgentNo = 999999;
```

El ejemplo 18.17 también demuestra varios aspectos de los subtipos y subtablas. Primero, el enunciado INSERT puede referirse a las columnas en ambos tipos debido a la relación de subtipo que involucra *ResidentialType* y *.PropertyType*. Segundo, insertar una fila en la tabla *Residential* inserta automáticamente una fila en la tabla *Property*, debido a la relación de subtabla entre las tablas *Residential* y *Property*.

Las columnas de referencia pueden actualizarse con el uso de un enunciado SELECT, de manera similar a la que se usó en el ejemplo 18.17. El ejemplo 18.18 demuestra un enunciado UPDATE que usa un enunciado SELECT para recuperar el identificador de objeto del agente relacionado.

**EJEMPLO 18.18 Uso de un enunciado SELECT para recuperar el identificador de objeto de la fila Agent relacionada**

```
UPDATE Residential
SET AgentRef =
( SELECT AgentOID FROM Agent WHERE AgentNo = 999998 )
WHERE PropNo = 999999;
```

Las expresiones de trayectoria que usan el operador punto (dot) y el operador dereference proporcionan una alternativa a los join tradicionales basados en valor en SQL-92. El ejemplo 18.19 muestra el uso de los operadores punto y dereference en expresiones de trayectoria. Para columnas con una fila o tipo definido por usuario, debe usar el operador punto en expresiones de trayectoria. La expresión *Address.City* hace referencia al componente Ciudad de la columna fila *Address*. Para columnas con un tipo de referencia, debe usar el operador dereference ( $\rightarrow$ ) en expresiones de trayectoria. La expresión *AgentRef->Name* recupera la columna *Name* de la fila relacionada *Agent*. Debe usarse el operador dereference ( $\rightarrow$ ) en lugar del operador punto porque la columna *AgentRef* tiene el tipo *REF(AgentType)*. La distinción entre los operadores punto y dereference es uno de los aspectos de mayor confusión en SQL:2003. Otros lenguajes orientados a objetos (como el lenguaje de consulta ODMG y Java) no tienen esta distinción.

#### **EJEMPLO 18.19 Enunciado SELECT con expresiones de trayectoria y el operador Dereference**

```
SELECT PropNo, P.Address.City, P.AgentRef->Address.City
  FROM Property P
 WHERE AgentRef->Name = 'John Smith'
```

En ocasiones, hay necesidad de probar la membresía en una tabla específica sin ser miembro de otras subtablas. El ejemplo 18.20 recupera propiedades industriales donde la columna pies cuadrados es mayor que 3 000. La cláusula FROM restringe el ámbito a las filas cuyo tipo más específico es *IndustrialType*. Por ende, el ejemplo 18.20 no recupera filas de la tabla *Residential*, una subtabla de la tabla *Property*.

#### **EJEMPLO 18.20 Uso de ONLY para restringir el rango de una tabla en una familia de subtablas**

```
SELECT PropNo, Address, Location
  FROM ONLY (Residential)
 WHERE Sqft > 1500
```

## 18.5 Características de base de datos de objetos en Oracle 10g

La parte de los paquetes de objetos SQL:2003 con mayor implementación es el tipo definido por usuario. Los grandes fabricantes de DBMS relacionales, incluidos IBM y Oracle, han implementado tipos definidos por usuario que ofrecen características similares al estándar SQL:2003. Los tipos definidos por usuario son importantes para almacenar y manipular datos complejos en bases de datos empresariales.

Las características de objeto en el estándar SQL:2003 están ganando aceptación comercial más allá de los tipos definidos por el usuario. A manera de ejemplo de implementación comercial de las características de objeto en SQL:2003, esta sección presenta las características de objeto más importantes de Oracle 10g con el uso de los ejemplos de las secciones anteriores. Aunque Oracle 10g no exige cumplimiento completo con las características de objeto, soporta la mayoría de las características de los paquetes de objeto SQL:2003, así como algunas características de objeto adicionales. A pesar de que no use Oracle 10g, puede ganar conocimiento acerca de la complejidad de las características de objeto de SQL:2003 y de la dificultad de asegurar conformidad mejorada con el estándar SQL:2003.

### 18.5.1 Definición de tipos definidos por el usuario y tablas tipo en Oracle 10g

Oracle 10g soporta tipos definidos por el usuario con una sintaxis cercana a la sintaxis SQL:2003. Como se muestra en el ejemplo 18.21, la mayoría de las diferencias son cosméticas, como las diferentes ubicaciones de los paréntesis, la palabra reservada RETURN en lugar de RETURNS en SQL:2003, y la palabra reservada OBJECT para tipos a nivel raíz. Las diferencias en los métodos son más significativas. Oracle 10g soporta funciones y procedimientos como métodos, en comparación con el único método de funciones en SQL:2003. Por ende, el procedimiento *Print* en el ejemplo 18.21 no se usa en el ejemplo 18.7, pues SQL:2003 no soporta el método de procedimientos.<sup>3</sup> Además, Oracle 10g soporta métodos de orden para realizar comparaciones objeto a objeto, métodos de mapeo para comparaciones indirectas de objetos y métodos estáticos para operaciones globales que no necesitan referencias a los datos de una instancia objeto.

#### EJEMPLO 18.21 Tipo *Point* en Oracle 10g

Este ejemplo corresponde al ejemplo 18.7 para SQL:2003.

```
CREATE TYPE Point AS OBJECT
(  x FLOAT(15),
   y FLOAT(15),
   MEMBER FUNCTION Distance(P2 Point) RETURN NUMBER,
   -- Calcula la distancia entre 2 puntos
   MEMBER FUNCTION Equals (P2 Point) RETURN BOOLEAN,
   -- Determina si 2 puntos son equivalentes
   MEMBER PROCEDURE Print  )
NOT FINAL
INSTANTIABLE;
```

Oracle 10g soporta herencia para tipos definidos por el usuario similares a SQL:2003. Una diferencia importante implica la invalidez de métodos. En Oracle 10g, invalidar métodos tiene el mismo nombre y firma en el tipo padre y el subtipo. Una firma consiste del nombre del método y el número, tipos y orden de los parámetros. Si dos métodos tienen firmas diferentes, no hay invalidación, ya que ambos métodos existen en el subtipo. Como en SQL:2003, la palabra clave OVERRIDING debe usarse cuando se invalide un método. En el ejemplo 18.22 no hay invalidación, ya que el método *Equals* en *ColorPoint* tiene una firma diferente a la del método *Equals* en *Point*. El método *Equals* en *ColorPoint* usa un argumento *ColorPoint*, mientras que el método *Equals* en *Point* usa un argumento *Point*. Sin embargo, el método *Print* en *ColorPoint* invalida el método *Print* en *Point*, pues ambos métodos tienen la misma firma.

Oracle 10g soporta tipos filas y tablas tipo de igual modo que SQL:2003, como se muestra en el ejemplo 18.23. De igual forma que SQL:2003, Oracle 10g soporta el tipo ROW y tipos definidos por el usuario para estructurar subconjuntos de columnas. Por ejemplo, en *AgentType*, el atributo de dirección podría usar el tipo ROW en lugar del *AddressType* definido por el usuario. Para el enunciado CREATE TABLE, Oracle 10g especifica identificadores de objeto de modo diferente que en SQL:2003. En Oracle 10g, la cláusula OBJECT IDENTIFIER define un identificador de objeto como generado por sistema o generado por usuario. Los identificadores

<sup>3</sup> La Guía de Desarrollo de Aplicaciones para Oracle 10g proporciona un ejemplo con el uso de paréntesis vacíos para procedimientos sin parámetros (*Print()*). Sin embargo, el compilador Oracle SQL despliega un mensaje de error cuando se incluyen los paréntesis. Por lo tanto, el procedimiento *Print* se declara sin paréntesis.

**EJEMPLO 18.22 Tipo ColorPoint en Oracle 10g**

Este ejemplo corresponde al ejemplo 18.8 para SQL:2003.

```
CREATE TYPE ColorPoint UNDER Point
(Color INTEGER,
 MEMBER FUNCTION Brighten (Intensity INTEGER) RETURN INTEGER,
 -- Aumenta intensidad de color
 MEMBER FUNCTION Equals (CP2 ColorPoint) RETURN BOOLEAN,
 -- Determina si 2 ColorPoints son equivalentes
 -- No se usa invalidar porque los dos métodos son equivalentes
 -- tienen firmas diferentes
OVERRIDING MEMBER PROCEDURE Print  )
NOT FINAL
INSTANTIABLE;
```

de objeto generados por sistema no tienen un nombre como el que proporciona SQL:2003. No obstante, Oracle ofrece funciones para manipular identificadores de objetos generados por sistema, de modo que no se necesita el nombre de una columna.

**EJEMPLO 18.23 Definición en Oracle 10g de AddressType y AgentType seguidos por la tabla tipo Agent con base en AgentType**

Este ejemplo corresponde al ejemplo 18.12 para SQL:2003.

```
CREATE TYPE AddressType AS OBJECT
( Street  VARCHAR(50),
 City    VARCHAR(30),
 State   CHAR(2),
 Zip     CHAR(9)  )
NOT FINAL;

CREATE TYPE AgentType AS OBJECT
(AgentNo  INTEGER,
 Name    VARCHAR(30),
 Address AddressType,
 Phone   CHAR(13),
 Email   VARCHAR(50)  )
NOT FINAL;

CREATE TABLE Agent OF AgentType
( CONSTRAINT AgentPK PRIMARY KEY(AgentNo)  )
OBJECT IDENTIFIER IS SYSTEM GENERATED ;
```

Oracle 10g soporta tipos de referencia para columnas del mismo modo que SQL:2003, como se muestra en el ejemplo 18.24. Sin embargo, el uso de la cláusula SCOPE es un poco diferente en Oracle 10g. En Oracle 10g, la cláusula SCOPE no puede usarse en un tipo definido por usuario como sí puede hacerse en SQL:2003. Para compensar, puede definirse una restricción de integridad referencial para limitar el ámbito de una referencia, como se muestra para la tabla *Property* en el ejemplo 18.24.

El ejemplo 18.25 muestra los tipos definidos por usuario para las propiedades residenciales e industriales, junto con las definiciones de tabla. Las cláusulas de restricción y de identificador de objeto se repiten en las tablas *Residential* e *Industrial* porque Oracle 10g no soporta subtablas.

**EJEMPLO 18.24 Definición en Oracle 10g de *.PropertyType* con una referencia a *AgentType* y la tabla tipo *Property***

Este ejemplo corresponde al ejemplo 18.13 para SQL:2003.

```
CREATE TYPE PropertyType AS OBJECT
(PropNo INTEGER,
Address AddressType,
SqFt INTEGER,
AgentRef REF AgentType,
Location Point )
NOT FINAL
INSTANTIABLE;

CREATE TABLE Property OF PropertyType
( CONSTRAINT PropertyPK PRIMARY KEY(PropNo),
CONSTRAINT AgentRefFK FOREIGN KEY(AgentRef) REFERENCES
Agent )
OBJECT IDENTIFIER IS SYSTEM GENERATED ;
```

**EJEMPLO 18.25 Enunciados *CREATE TYPE* y *CREATE TABLE* para propiedad residencial e industrial**

Este ejemplo corresponde al ejemplo 18.14 para SQL:2003.

```
CREATE TYPE AssessType AS VARRAY(6) OF DECIMAL(9,2);

CREATE TYPE ResidentialType UNDER PropertyType
(BedRooms INTEGER,
BathRooms INTEGER,
Assessments AssessType )
NOT FINAL
INSTANTIABLE;

CREATE TABLE Residential OF ResidentialType
(CONSTRAINT ResidentialPK PRIMARY KEY(PropNo),
CONSTRAINT AgentRefFK1 FOREIGN KEY(AgentRef) REFERENCES
Agent )
OBJECT IDENTIFIER IS SYSTEM GENERATED ;

CREATE TYPE IndustrialType UNDER PropertyType
(Zoning VARCHAR(20),
AccessDesc VARCHAR(20),
RailAvailable CHAR(1),
Parking VARCHAR(10) )
NOT FINAL
INSTANTIABLE;

CREATE TABLE Industrial OF IndustrialType
(CONSTRAINT IndustrialPK PRIMARY KEY(PropNo),
CONSTRAINT AgentRefFK2 FOREIGN KEY(AgentRef) REFERENCES
Agent )
OBJECT IDENTIFIER IS SYSTEM GENERATED ;
```

El ejemplo 18.25 también muestra diferencias entre la declaración de columnas de arreglo en Oracle 10g y SQL:2003. En Oracle 10g, el constructor VARRAY no puede usarse directamente con columnas de una tabla o atributos de un tipo definido por el usuario. En vez de ello, el constructor VARRAY debe usarse en un tipo separado definido por el usuario, como se muestra en el tipo *AssessType* del ejemplo 18.25. Además, Oracle 10g usa paréntesis para el tamaño del arreglo en lugar de los corchetes utilizados en SQL:2003.

### 18.5.2 Uso de tablas de tipos en Oracle 10g

Esta sección comienza con enunciados de manipulación para insertar y modificar objetos en las tablas tipo. El ejemplo 18.26 demuestra un enunciado INSERT que usa un nombre de tipo para la columna estructurada *Address*. Si la columna *Address* se definió con el constructor tipo ROW, la sintaxis Oracle 10g sería idéntica al ejemplo 18.15 con la palabra clave ROW en sustitución de *AddressType*.

#### EJEMPLO 18.26 Inserción de dos fila en la tabla tipo Agent

Este ejemplo corresponde al ejemplo 18.16 para SQL:2003.

```
INSERT INTO Agent
(AgentNo, Name, Address, Email, Phone)
VALUES (999999, 'Sue Smith',
        AddressType('123 Any Street', 'Denver', 'CO', '80217'),
        'sue.smith@anyisp.com', '13031234567);

INSERT INTO Agent
(AgentNo, Name, Address, Email, Phone)
VALUES (999998, 'John Smith',
        AddressType('123 Big Street', 'Boulder', 'CO', '80217'),
        'john.smith@anyisp.com', '13034567123');
```

Debido a que Oracle 10g no soporta subtablas, se usan enunciados de manipulación adicionales para simular inclusión de conjunto entre subtablas. En el ejemplo 18.27, los enunciados INSERT se usan tanto en la tabla padre como en la subtabla. Lo ideal sería poder definir disparadores para ocultar los enunciados de manipulación adicionales.

#### EJEMPLO 18.27 Enunciados INSERT para agregar un objeto en las tablas Property y Residential

Este ejemplo corresponde al ejemplo 18.17 para SQL:2003.

```
INSERT INTO Residential
(PropNo, Address, SqFt, AgentRef, BedRooms, BathRooms, Assessments)
SELECT 999999, AddressType('123 Any Street', 'Denver', 'CO', '80217'),
       2000, REF(A), 3, 2, AssessType(190000, 200000)
FROM Agent A
WHERE AgentNo = 999999;

-- Este enunciado INSERT conserva inclusión de conjunto entre las tablas Property
-- y las tablas Residential.
INSERT INTO Property
(PropNo, Address, SqFt, AgentRef)
SELECT 999999, AddressType('123 Any Street', 'Denver', 'CO', '80217'),
       2000, REF(A)
FROM Agent A
WHERE AgentNo = 999999;
```

El ejemplo 18.27 también demuestra la función REF para obtener un identificador de objeto generado por sistema. Cuando se usa la función REF, debe usarse una variable de correlación como el parámetro (alias de tabla). No puede usar el nombre de tabla en lugar de la variable de correlación. El enunciado REF también puede usarse en enunciados UPDATE, como se demuestra en el ejemplo 18.28.

**EJEMPLO 18.28****Uso de un enunciado SELECT con la función REF para recuperar el identificador de objeto de la fila relacionada Agent**

Este ejemplo corresponde al ejemplo 18.18 para SQL:2003.

```
UPDATE Residential
  SET AgentRef =
    ( SELECT REF(A) FROM Agent A WHERE AgentNo = 999998 )
    WHERE PropNo = 999999;

-- Este enunciado UPDATE conserva consistencia entre las tablas Property
-- y las tablas Residential.
UPDATE Property
  SET AgentRef =
    ( SELECT REF(A) FROM Agent A WHERE AgentNo = 999998 )
    WHERE PropNo = 999999;
```

Oracle 10g soporta expresiones de trayectoria que contengan el operador punto y la función DEREF. La función DEREF también puede usarse en SQL:2003 en lugar del operador  $\rightarrow$ . La función DEREF usa un identificador de objeto como parámetro, como se muestra en el ejemplo 18.29. Cuando se usan columnas que tengan un tipo de objeto como *Address*, debe usarse una variable de correlación.

**EJEMPLO 18.29****Enunciado SELECT de Oracle 10g con expresiones de trayectoria que contienen el operador punto y la función DEREF**

Este ejemplo corresponde al ejemplo 18.19 para SQL:2003.

```
SELECT PropNo, P.Address.City, DEREF(AgentRef).Address.City
  FROM Property P
 WHERE DEREF(AgentRef).Name = 'John Smith';
```

Aunque Oracle 10g soporta la función DEREF, parece no ser necesario su uso. El operador punto puede usarse en expresiones de trayectoria, incluso cuando una columna tiene un tipo de referencia, como se muestra en el ejemplo 18.30. Note que es necesaria una variable de correlación cuando se usan columnas REF en una expresión de trayectoria con el operador punto.

**EJEMPLO 18.30****Enunciado SELECT de Oracle 10g con expresiones de trayectoria que contienen el operador punto en lugar de la función DEREF**

Este ejemplo corresponde al ejemplo 18.19 para SQL:2003.

```
SELECT PropNo, P.Address.City, P.AgentRef.Address.City
  FROM Property P
 WHERE P.AgentRef.Name = 'John Smith';
```

Al igual que SQL:2003, Oracle 10g soporta la palabra clave ONLY en la cláusula FROM. Sin embargo, en Oracle 10g la palabra clave ONLY se aplica a vistas, no a tablas. Por ende, el ejemplo 18.20 no funcionará en Oracle 10g, a menos que, en lugar de tablas, se usen vistas de objeto.

La función VALUE toma una variable de correlación como un parámetro y emite instancias de la tabla de objetos asociada con la variable de correlación. Por lo tanto, puede usarse la función VALUE para recuperar todas las columnas de tablas tipo en lugar de usar \* para tablas no tipo, como se muestra en el ejemplo 18.31. No obstante, Oracle 10g proporciona un mensaje de error para el enunciado del ejemplo 18.31 porque la función VALUE no funciona en tablas basadas en tipos no finales. El enunciado SELECT del ejemplo 18.32 funciona correctamente porque *AgentType1* carece de la cláusula NOT FINAL.

#### **EJEMPLO 18.31 Uso de la Función VALUE para recuperar todas las columnas de una tabla tipo**

Note que este enunciado causa un error en Oracle 10g porque *AgentType* no es final.

```
SELECT VALUE(A) FROM Agent A;
```

#### **EJEMPLO 18.32 Uso de la función VALUE para recuperar todas las columnas de una tabla basada en un tipo final**

```
CREATE TYPE AgentType1 AS OBJECT
(AgentNo INTEGER,
 Name  VARCHAR(30),
 Phone CHAR(10),
 Email  VARCHAR(50) );
CREATE TABLE Agent1 OF AgentType1
( CONSTRAINT Agent1PK PRIMARY KEY(AgentNo) )
OBJECT IDENTIFIER IS SYSTEM GENERATED ;
INSERT INTO Agent1
(AgentNo, Name, Email, Phone)
VALUES (999998, 'John Smith',
 'john.smith@anyisp.com', '3037894567');
SELECT VALUE(A) FROM Agent1 A;
```

### **18.5.3 Otras características de objeto en Oracle 10g**

Oracle 10g ofrece características adicionales de objeto, algunas de las cuales extienden las características de objetos en SQL:2003. La capacidad de sustituir tipos y las vistas jerárquicas ofrecen alternativas limitadas para las subtablas. El tipo de colección TABLE, correspondiente al tipo multiset de SQL:2003, soporta tablas anidadas. Oracle XML DB proporciona almacenamiento y manipulación eficientes de grandes depósitos de documentos XML. Esta sección ofrece un panorama de dichas características de objetos. Para más detalles acerca de estas características de objetos, debe consultar la documentación en línea de Oracle 10g.

#### *Sustitución de tipo y vistas jerárquicas*

La documentación de Oracle 10g sugiere el uso de la capacidad de sustitución de tipos para administrar extensiones de tablas padre y subtablas relacionadas. La capacidad de sustitución de tipos significa que una columna o fila definida como de tipo X puede contener instancias de X y cualesquiera de sus subtipos. Cuando se usa sustitución de tipo para soportar subtablas, los tipos definidos por usuario se definen con relaciones de subtipo, pero sólo se define una tabla tipo (la tabla raíz). Para las relaciones de inclusión de conjunto, todas las operaciones de manipulación se realizan en la tabla raíz con el uso de la sustitución de tipos. Sin embargo, no es clara la sintaxis para usar tipos de referencia y columnas de subtipo en enunciados de manipulación. Ciertos enunciados INSERT y UPDATE no funcionan con tipos sustituidos. Para administrar

relaciones de inclusión de conjunto, la sustitución de tipo no soporta traslapamiento de subtipos. Por ende, la sustitución de tipos no proporciona una solución satisfactoria para subtablas debido a la sintaxis limitada y el soporte incompleto para relaciones de inclusión de conjuntos.

La documentación de Oracle 10g también sugiere el uso de vistas jerárquicas para gestionar extensiones de tablas padre y subtablas relacionadas. Las vistas jerárquicas usan la palabra clave UNDER, tal como las jerarquías de tipos. A diferencia de las familias de subtablas, para almacenar vistas jerárquicas son posibles muchos modelos de almacenamiento. El administrador de bases de datos debe elegir el mejor modelo de almacenamiento y asegurar que los usuarios entiendan cómo usar las vistas jerárquicas en los enunciados de recuperación y manipulación. Aunque las vistas jerárquicas son útiles, no ofrecen un sustituto satisfactorio para las subtablas.

### *Tablas anidadas*

Oracle 10g proporciona soporte extenso para múltiples niveles de tablas anidadas correspondientes a la característica multiset de SQL:2003. Como se afirmó anteriormente, el uso de tablas anidadas no es claro para bases de datos empresariales. Hasta que la teoría y la práctica ofrezcan un mayor conocimiento, el uso de tablas anidadas será destinado para situaciones especializadas. La siguiente lista resume el soporte de Oracle para las tablas anidadas con el fin de indicar sus características:

- Los constructores de tipo TABLE y NESTED TABLE soportan enunciados CREATE TABLE con tablas anidadas. Puede especificarse un tipo definido por usuario con el constructor TABLE y luego usarlo en un enunciado CREATE TABLE con el constructor NESTED TABLE.
- Las columnas con tipos de tabla anidada pueden aparecer en los resultados de la consulta. El operador TABLE aplana las tablas anidadas si un usuario quiere ver resultados planos en lugar de anidados.
- Los operadores de comparación soportan comparaciones de igualdad, subconjunto y membresía entre tablas anidadas.
- Los operadores de conjunto soportan las operaciones unión, intersección y diferencia sobre las tablas anidadas, así como remoción de duplicado y cardinalidad de tabla anidada.
- Las vistas de objeto soportan niveles múltiples de tablas anidadas.

### *Soporte de documentos XML*

El Lenguaje de Mercado eXtensible (XML) ha surgido como fundamento para los negocios electrónicos para consumidores y organizaciones. XML es un metalenguaje que soporta la especificación de otros lenguajes. Para restringir documentos XML, pueden definirse esquemas XML. Un esquema XML especifica la estructura, contenido y significado de un conjunto de documentos XML. Los esquemas XML soportan intercambio mejorado de datos, búsqueda en Internet y calidad de datos. Muchos dominios de aplicación han desarrollado esquemas XML como elemento esencial del comercio electrónico.

Como resultado de la creciente importancia de XML, el soporte para almacenamiento y manipulación de documentos XML se ha vuelto una prioridad para los DBMSs. La parte 14 de SQL:2003 se dedica a almacenamiento y manipulación de documentos XML. Oracle y otros fabricantes de DBMS comerciales han dedicado una gran cantidad de investigación y desarrollo para soportar la especificación de la parte 14 y características adicionales. La característica más prominente es un nuevo tipo de datos XML que la mayoría de los proveedores DBMS soportan como un tipo de datos preconstruido. Para ofrecer una idea acerca del soporte XML extenso disponible en los DBMS comerciales, la siguiente lista resume sus características en Oracle XML DB.

- El tipo de datos XMLType permite que los documentos XML se almacenen como tablas y columnas de una tabla.
- Las variables en los procedimientos PL/SQL pueden usar el tipo de datos XMLType. Una interfaz de programación de aplicación para XMLType soporta todo un rango de operaciones sobre documentos XML.

- Los documentos XML pueden almacenarse en un formato estructurado utilizando el tipo de datos XMLType, o en un formato no estructurado con el uso del tipo CLOB. El almacenamiento como datos XMLType permite indexado y optimización de consulta especializada.
- El esquema XML soporta aplicaciones tanto a documentos XML como a tablas relacionales. Oracle puede forzar restricciones en un esquema XML tanto a tablas como a documentos XML almacenados en una base de datos.
- La dualidad XML/SQL permite que los mismos datos se manipulen como tablas o como documentos XML. Los datos relacionales pueden convertirse en XML y desplegarse como HTML. Los documentos XML pueden convertirse en datos relacionales.
- Oracle soporta la mayoría de los operadores XML en el estándar SQL:2003. En particular, Oracle soporta los operadores de recorrido XML existsNode(), extract(), extractValue(), updateXML() y XMLSequence() en el estándar SQL/XML.
- La reescritura de consulta transforma las operaciones que recorren los documentos XML en enunciados SQL estándar. El optimizador de consulta de Oracle procesa la reescritura del enunciado SQL en la misma forma que otros enunciados SQL.

## Reflexión final

Este capítulo describió la motivación, principios y arquitecturas de los DBMS de objetos. La tecnología de bases de datos de objetos es impulsada por la necesidad de integrar datos complejos y simples y por problemas de productividad de software debido a incompatibilidades de tipo entre DBMS y lenguajes de programación. Se explicaron y relacionaron con el desarrollo de los DBMS de objetos tres principios de la computación orientada a objetos: encapsulación, herencia y polimorfismo. La encapsulación (ocultamiento de los detalles de implementación) soporta la independencia de datos. La encapsulación usualmente es relajada para que los DBMS permitan consultas *ad hoc*. La herencia (compartir código y datos) soporta reusabilidad de software. El polimorfismo (admitir múltiples implementaciones para procedimientos) permite modificación incremental y un menor vocabulario de procedimientos.

Debido a la diversidad de formas para implementar los principios orientados a objetos y su dificultad de implementación, están disponibles comercialmente algunas arquitecturas DBMS de objetos. Este capítulo describió cinco arquitecturas y discutió sus ventajas y desventajas. Las primeras dos arquitecturas no soportan por completo los principios orientados a objetos, ya que implican esquemas simples para almacenar objetos grandes e invocan servidores especializados afuera de un DBMS. Las últimas tres arquitecturas proporcionan trayectorias diferentes para implementar los principios orientados a objetos. Las arquitecturas difieren principalmente en el nivel de integración con el DBMS. El middleware de base de datos de objetos implica la última integración con un DBMS, pero ofrece el ámbito más amplio de datos complejos. Los DBMS relacionales de objetos contienen un procesador de consulta de objeto en lo alto de un kernel relacional. Los DBMS orientados a objetos contienen tanto un procesador de consulta de objetos como un kernel de objetos. Con la creciente aceptación de las características relacionales de objetos en SQL:2003, el mercado se ve fuertemente favorecido por el enfoque relacional de objetos.

Para proporcionar una visión más concreta de las bases de datos de objetos, este capítulo presentó la definición de base de datos de objetos y las características de manipulación de SQL:2003. Los tipos definidos por usuario soportan nuevos tipos de datos complejos. Las expresiones en las consultas pueden hacer referencia a columnas con base en tipos definidos por usuario y el uso de métodos de tipos definidos por usuario. SQL:2003 soporta la herencia y el polimorfismo para tipos definidos por el usuario, así como para relaciones de inclusión de conjunto para familias de subtablas. Debido a la complejidad de SQL:2003, existen pocos DBMSs que soporten la concordancia mejorada del estándar. Las características relacionales de objeto de Oracle 10g se presentaron con el fin de demostrar la implementación de muchas características de objeto SQL:2003.

## Revisión de conceptos

- Ejemplos de datos complejos que pueden almacenarse en formato digital.
- Uso de datos complejos como motivación para la tecnología de base de datos de objetos.
- Incompatibilidades de tipo como motivación para la tecnología de base de datos de objetos.
- Encapsulación como un tipo de independencia de datos.
- Relajamiento de la encapsulación para soportar consultas *ad hoc*.
- Herencia como forma de compartir código y datos.
- Polimorfismo para reducir el vocabulario de los procedimientos y permitir la compartición incremental de código.
- Diferencia entre ligadura estática y dinámica.
- Comprobación de tipo fuerte para eliminar errores de incompatibilidad en expresiones.
- Razones para la diversidad de arquitecturas de gestión en las bases de datos de objetos.
- Nivel de integración con un DBMS para cada arquitectura de gestión de base de datos de objeto.
- Características de cada arquitectura de administración de base de datos de objetos.
- Razones para la falta de éxito de los DBMS de conformidad con ODMG.
- Tipos definidos por usuario en SQL:2003 para definir datos complejos y operaciones.
- Familias de subtabla en SQL:2003: herencia e inclusión de conjunto.
- Relación de familias de subtabla y tipos definidos por el usuario.
- Uso del tipo fila de SQL:2003 y tipo de referencia en tablas de objeto.
- Uso de expresiones de trayectoria y el operador dereference ( $\rightarrow$ ) en enunciados SELECT de SQL:2003.
- Subtablas referenciadas en enunciados SELECT.
- Definición y uso de tipos definidos por el usuario y tablas tipo en Oracle 10g.
- Diferencias entre características de objeto entre Oracle 10g y SQL:2003.

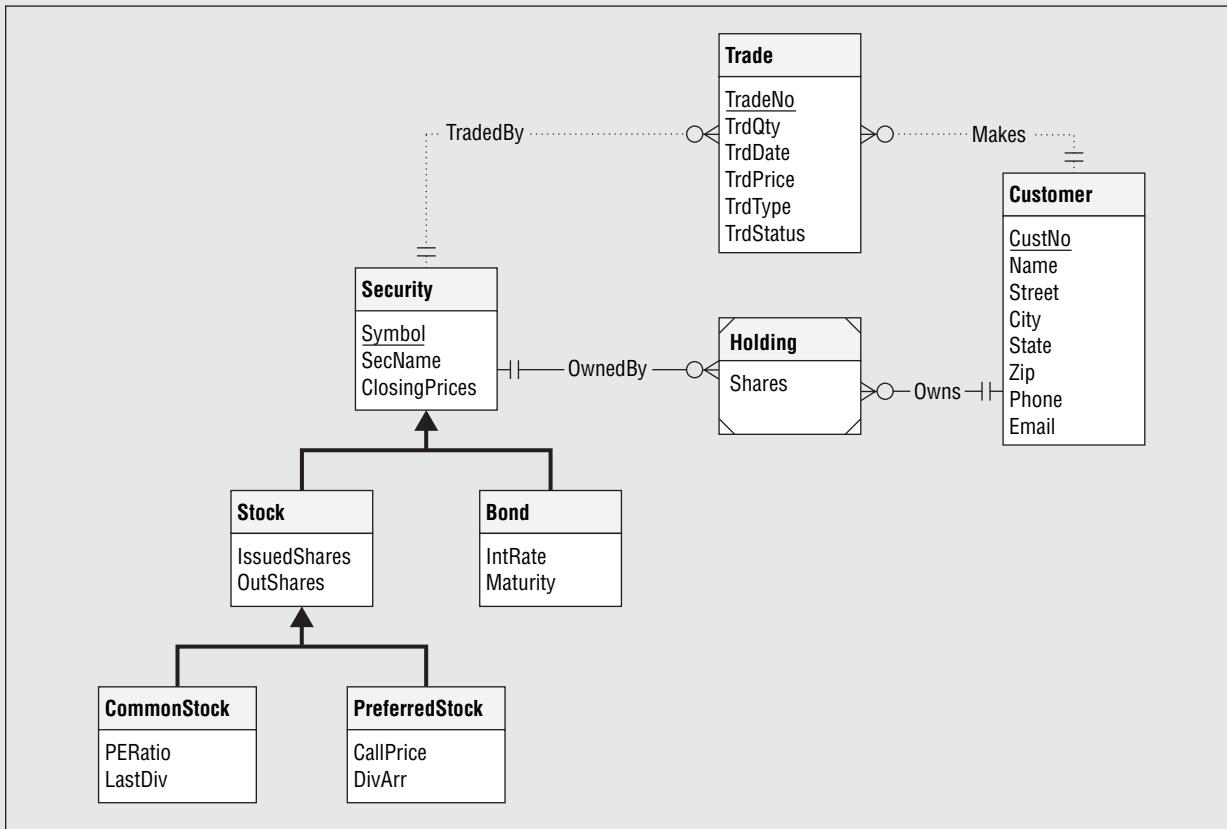
## Preguntas

1. ¿Cómo impulsa la necesidad de la tecnología de bases de datos de objetos el uso de datos complejos?
2. ¿Qué problemas causan las incompatibilidades entre los tipos proporcionados por un DBMS y un lenguaje de programación?
3. Presente un ejemplo de aplicación que use tanto datos simples como complejos. Use una aplicación diferente a la discutida en la sección 18.1.3.
4. Defina encapsulación. ¿Cómo soporta la encapsulación la meta de independencia de datos?
5. ¿Cómo relaja la encapsulación el DBMS de objetos? ¿Por qué se relaja la encapsulación? ¿Cuál es el papel del control de seguridad de las bases de datos como sustituto para la encapsulación?
6. Defina herencia. ¿Cuáles son los beneficios de la herencia?
7. Defina polimorfismo. ¿Cuáles son los beneficios del polimorfismo?
8. ¿Qué es comprobación de tipo fuerte? ¿Por qué es importante la comprobación de tipo fuerte?
9. ¿Cuál es la diferencia entre ligadura estática y dinámica?
10. ¿Qué implementación de principios orientados a objeto ocurrió primero: los lenguajes de programación orientados a objetos o los DBMS de objetos?
11. Compare el énfasis en los lenguajes de programación orientados a objetos con los DBMSs de objetos.
12. Discuta las razones por las que han sido desarrolladas múltiples arquitecturas de DBMS de objeto.
13. Discuta los beneficios y limitaciones de almacenar objetos grandes en las bases de datos. ¿Por qué se necesita un software externo cuando se almacenan objetos grandes en las bases de datos?
14. Discuta los beneficios y limitaciones de usar servidores de medios especializados.
15. Discuta los beneficios y limitaciones de usar middleware de base de datos de objetos. ¿Por qué el middleware de base de datos de objetos soporta el más amplio rango de datos complejos?

16. Discuta los beneficios y limitaciones de usar un DBMS relacional de objetos. ¿Qué cambios se hacen al procesador de consulta de un DBMS relacional para convertirlo en un DBMS relacional de objetos?
17. Discuta los beneficios y limitaciones de usar un DBMS orientado a objetos. ¿Cómo difiere un DBMS orientado a objetos de un DBMS relacional de objetos?
18. ¿Cuál es el estatus del estándar ODMG y de los DBMS que se apegan a ODMG?
19. ¿Qué arquitecturas DBMS de objeto cree que serán las dominantes dentro de cinco años?
20. ¿Qué es un objeto persistente? ¿Cuál es la diferencia entre un objeto persistente y un objeto temporal?
21. ¿Por qué hay tres lenguajes estándar para los DBMSs de objetos?
22. ¿Cuáles son los componentes de un tipo definido por el usuario en SQL:2003?
23. ¿Cuáles son las diferencias entre los métodos, funciones y procedimientos de SQL:2003?
24. ¿Cómo se usan los tipos definidos por el usuario de SQL:2003 en las definiciones y expresiones de tablas?
25. ¿Qué es un tipo fila? ¿Cómo se usan los tipos fila en las definiciones de tabla de SQL:2003?
26. Explique las diferencias de encapsulación para tipos definidos por el usuario versus tablas tipo en SQL:2003.
27. ¿Qué es una tabla tipo?
28. ¿Cómo define una subtabla?
29. Discuta la relación de las familias de subtabla e inclusión de conjunto.
30. ¿Qué efectos colaterales ocurren cuando se inserta una fila en una subtabla?
31. ¿Qué efectos colaterales ocurren cuando se actualiza una fila en una subtabla?
32. ¿Qué efectos colaterales ocurren cuando se borra una fila en una subtabla?
33. ¿Cuál es la diferencia entre una llave foránea y una referencia?
34. ¿Cuándo debe usarse un enunciado SELECT como parte de un enunciado INSERT al agregar objetos a una tabla tipo?
35. ¿Cuál es la diferencia de notación entre tablas de combinación que se vinculan mediante una llave foránea versus vinculación a una columna con un tipo referencia?
36. ¿Qué es una expresión de trayectoria? ¿Cuándo usa una expresión de trayectoria?
37. ¿Cuándo necesita usar el operador dereference ( $\rightarrow$ ) en una expresión de trayectoria?
38. ¿Cuál es el propósito de la palabra clave ONLY en un enunciado SELECT de SQL:2003?
39. Compare y contraste métodos en SQL:2003 con métodos en Oracle 10g.
40. ¿Cuáles son los criterios para invalidar un método en Oracle 10g?
41. ¿Cuál es la limitación más significativa para bases de datos de objetos en Oracle 10g, en comparación con SQL:2003?
42. Discuta brevemente la importancia de las características de objeto en Oracle 10g que no son parte de SQL:2003.
43. ¿Qué es conformidad mínima para SQL:2003?
44. ¿Qué es conformidad mejorada para SQL:2003?
45. Discuta brevemente el estado de prueba de conformidad para SQL:2003.
46. En SQL:2003, ¿cuál es la diferencia entre los tipos de colección ARRAY y MULTISET?
47. ¿Cuáles son las contrapartes Oracle 10g de los tipos de colección SQL:2003?
48. ¿Cuál es el papel de las tablas anidadas en el diseño de tablas y el desarrollo de aplicaciones de bases de datos?
49. ¿Cuáles son los tipos de datos preconstruidos definidos por el usuario que se encuentran disponibles comercialmente en los DBMSs del mercado?
50. ¿Qué es un paquete en el estándar SQL:2003?

## Problemas

Los problemas ofrecen práctica en el uso de SQL:2003 y Oracle 10g para definir tipos definidos por el usuario y tablas tipo, así como para el uso de tablas tipo. Los problemas del 1 al 26 involucran SQL:2003, mientras que los problemas del 27 al 52 involucran a Oracle 10g. Si no tiene acceso a Oracle 10g, puede usar Oracle 9i o IBM DB2. Los problemas involucran la base de datos financiera que se muestra en la figura 18.P1.

**FIGURA 18.P1** ERD para la base de datos financiera**TABLA 18.P1**  
Lista de métodos para el tipo TimeSeries

Nombre	Parámetros	Resultado
WeeklyAvg	TimeSeries	TimeSeries
MonthlyAvg	TimeSeries	TimeSeries
YearlyAvg	TimeSeries	TimeSeries
MovingAvg	TimeSeries, Start Date, Number of Values	Float
RetrieveRange	TimeSeries, Start Date, Number of Values	TimeSeries

- Utilizando SQL:2003, defina un tipo definido por usuario para una serie de tiempo. Las variables de una serie de tiempo incluyen un arreglo de valores de punto flotante (máximo de 365), fecha de inicio, duración (número máximo de puntos de datos en la serie de tiempo), tipo de calendario (personal o de negocios) y periodo (día, semana, mes o año). Defina métodos como los mencionados en la tabla 18.P1. Necesita definir los parámetros para los métodos, no el código para implementar los métodos. El parámetro *TimeSeries* se refiere al objeto implícito *TimeSeries*.
- Utilizando SQL:2003, defina un tipo valor y una tabla tipo para valor. Un valor tiene campos para el símbolo único, el nombre del valor y una serie de tiempo de precios de cierre. Tanto el tipo valor como la tabla no tienen padre.
- Utilizando SQL:2003, defina un tipo acción y una tabla tipo para acción. Una acción tiene campos para el número de participaciones emitidas, el número de participaciones destacables y la serie de tiempo de precios de cierre. La tabla acción hereda de la tabla valor, y el tipo acción hereda del tipo valor.
- Utilizando SQL:2003, defina un tipo bono y una tabla tipo para bono. Un bono tiene campos para la tasa de interés y la fecha de vencimiento. La tabla bono hereda de la tabla valor, y el tipo bono hereda del tipo valor.

5. Utilizando SQL:2003, defina un tipo acción común y una tabla tipo común para acción. Una acción común tiene campos para la relación precio-utilidades y el último importe de dividendo. La tabla acción común hereda de la tabla acción, y el tipo acción común hereda del tipo acción común.
6. Utilizando SQL:2003, defina un tipo acción preferencial y una tabla preferencial tipo acción. Una acción preferencial tiene campos para el precio nominal y dividendos en retraso. La tabla acción preferencial hereda de la tabla acción, y el tipo acción preferencial hereda del tipo acción común.
7. Utilizando SQL:2003, defina un tipo cliente y una tabla tipo para cliente. Un cliente tiene campos para número único de cliente, nombre, dirección, teléfono y dirección de correo electrónico. El campo dirección es un tipo fila con campos para calle, ciudad, estado y código postal. El campo teléfono es un tipo fila con campos para código de país, código de área y número local. Debe definir tipos para dirección y teléfono, de modo que los tipos puedan reutilizarse. Tanto el tipo como la tabla de cliente no tienen padre.
8. Utilizando SQL:2003, defina un tipo cartera de inversión y una tabla tipo para cartera de inversión. Una inversión tiene campos para cliente (tipo de datos referencia), valores (tipo de datos referencia) y las participaciones poseídas. La llave primaria de la tabla *Holding* es una combinación del campo *CustNo* del cliente relacionado y el campo *Symbol* del valor relacionado. Defina la integridad referencial o restricciones SCOPE para limitar el rango de la referencia cliente y la referencia valor. Tanto el tipo como la tabla inversión no tienen padre.
9. Utilizando SQL:2003, defina un tipo comercio y una tabla tipo para comercio. Comercio tiene campos para número único de comercio, cliente (tipo dato referencia), valor (tipo dato referencia), fecha de comercio, cantidad, precio unitario, tipo (compra o venta) y estatus (pendiente o completo). La llave primaria de la tabla *Trade* es el número de comercio. Defina la integridad referencial o restricciones de SCOPE para limitar el rango de la referencia cliente y la referencia valor. Tanto el tipo como la tabla comercio no tienen padre.
10. Utilizando SQL:2003, inserte un objeto en la tabla tipo *CommonStock* para acciones comunes de Microsoft.
11. Utilizando SQL:2003, inserte un objeto en la tabla tipo *CommonStock* para acciones comunes de Dell Corporation.
12. Utilizando SQL:2003, inserte un objeto en la tabla tipo *CommonStock* para acciones comunes de IBM. Ingrese un valor en la columna precios de cierre (tipo serie de tiempo) mediante la especificación del arreglo de valores, el periodo, el tipo de calendario, la fecha de inicio y la duración.
13. Utilizando SQL:2003, inserte un objeto en la tabla tipo *Bond* para un bono corporativo IBM.
14. Utilizando SQL:2003, inserte un objeto en la tabla tipo *Customer*. Use 999999 como el número de cliente, John Smith como el nombre de cliente y Denver como la ciudad.
15. Utilizando SQL:2003, inserte un objeto en la tabla tipo *Customer*. Use 999998 como el número de cliente, Sue Smith como el nombre de cliente y Boulder como la ciudad.
16. Utilizando SQL:2003, inserte un objeto en la tabla tipo *Holding*. Conecte el objeto de cartera de inversión con el objeto *Security* de Microsoft y el objeto *Customer* de Sue Smith. Use 200 como el número de participaciones poseídas.
17. Utilizando SQL:2003, inserte un objeto en la tabla tipo *Holding*. Conecte el objeto de cartera de inversión con el objeto *Secutiry* de IBM y el objeto *Customer* de Sue Smith. Use 100 como el número de participaciones poseídas.
18. Utilizando SQL:2003, inserte un objeto en la tabla tipo *Trade*. Conecte el objeto comercio con el objeto acción común de IBM y el objeto *Customer* de Sue Smith. Use 100 como la cantidad de participaciones negociadas, “comprar” como el tipo de comercio y otros valores de su elección para las otras columnas.
19. Utilizando SQL:2003, inserte un objeto en la tabla tipo *Trade*. Conecte el objeto comercio con el objeto acción común de Microsoft y el objeto *Customer* de Sue Smith. Use 200 como la cantidad de participaciones negociadas, “comprar” como el tipo de comercio y otros valores de su elección para las otras columnas.
20. Utilizando SQL:2003, inserte un objeto en la tabla tipo *Trade*. Conecte el objeto comercio con el objeto bono corporativo de IBM y el objeto *Customer* de John Smith. Use 150 como la cantidad de participaciones negociadas, “comprar” como el tipo de comercio y otros valores de su elección para las otras columnas.
21. Utilizando SQL:2003, actualice la columna de referencia de cliente del objeto *Holding* del problema 17 al objeto *Customer* de John Smith.

22. Utilizando SQL:2003, actualice la columna de referencia de cliente del objeto *Trade* del problema 19 al objeto *Customer* de John Smith.
23. Utilizando SQL:2003, escriba un enunciado SELECT para listar los valores poseídos por clientes de Denver. Sólo enliste los valores con más de 100 participaciones poseídas. Incluya en el resultado nombre de cliente, símbolo y participaciones poseídas.
24. Utilizando SQL:2003, escriba un enunciado SELECT para listar los seguros comprados por clientes de Boulder. Incluya en el resultado nombre de cliente, símbolo del seguro, número de comercio, fecha de comercio, cantidad comercializada y precio unitario.
25. Utilizando SQL:2003, escriba un enunciado SELECT para listar nombre de cliente, símbolo de valores y precios de cierre para cada acción poseída por clientes de Denver.
26. Utilizando SQL:2003, escriba un enunciado SELECT para listar nombre de cliente, símbolo de valores, número de comercio, fecha de comercio, cantidad comercializada y precio unitario para compras de acciones comunes por clientes de Boulder.
27. Utilizando Oracle 10g, defina un tipo definido por usuario para una serie de tiempo. Las variables de una serie de tiempo incluyen un arreglo de valores de punto flotante (máximo de 365), fecha de inicio, duración (número máximo de puntos de datos en la serie de tiempo), tipo de calendario (personal o de negocios) y periodo (día, semana, mes o año). Defina métodos como los mencionados en la tabla 18.P1. Necesita definir los parámetros para los métodos, no el código para implementar los métodos. El parámetro *TimeSeries* se refiere al objeto implícito *TimeSeries*.
28. Utilizando Oracle 10g, defina un tipo valor y una tabla tipo para valor. Un valor tiene campos para el símbolo único, el nombre del valor y una serie de tiempo de precios de cierre. Tanto el tipo valor como la tabla no tienen padre.
29. Utilizando Oracle 10g, defina un tipo acción y una tabla tipo para acción. Una acción tiene campos para el número de participaciones emitidas, el número de participaciones destacables y la serie de tiempo de precios de cierre. La tabla acción hereda de la tabla valor, y el tipo acción hereda del tipo valor.
30. Utilizando Oracle 10g, defina un tipo bono y una tabla tipo para bono. Un bono tiene campos para la tasa de interés y la fecha de vencimiento. La tabla bono hereda de la tabla valor, y el tipo bono hereda del tipo valor.
31. Utilizando Oracle 10g, defina un tipo acción común y una tabla tipo común para acción. Una acción común tiene campos para la relación precio-utilidades y el último importe de dividendo. La tabla acción común hereda de la tabla acción, y el tipo acción común hereda del tipo acción común.
32. Utilizando Oracle 10g, defina un tipo acción preferencial y una tabla preferencial tipo acción. Una acción preferencial tiene campos para el precio nominal y dividendos en mora. La tabla acción preferencial hereda de la tabla acción, y el tipo acción preferencial hereda del tipo acción común.
33. Utilizando Oracle 10g, defina un tipo cliente y una tabla tipo para cliente. Un cliente tiene campos para número único de cliente, nombre, dirección, teléfono y dirección de correo electrónico. El campo dirección es un tipo fila con campos para calle, ciudad, estado y código postal. El campo teléfono es un tipo fila con campos para código de país, código de área y número local. Debe definir tipos para dirección y teléfono, de modo que los tipos puedan reutilizarse. Tanto el tipo como la tabla de cliente no tienen padre.
34. Utilizando Oracle 10g, defina un tipo cartera de inversión y una tabla tipo para cartera de inversión. Una inversión tiene campos para cliente (tipo de datos referencia), valores (tipo de datos referencia) y las participaciones poseídas. La llave primaria de la tabla *Holding* es una combinación del campo *CustNo* del cliente relacionado y el campo *Symbol* del valor relacionado. Defina la integridad referencial o restricciones SCOPE para limitar el rango de la referencia cliente y la referencia valor. Tanto el tipo como la tabla inversión no tienen padre.
35. Utilizando Oracle 10g, defina un tipo comercio y una tabla tipo para comercio. Comercio tiene campos para número único de comercio, cliente (tipo dato referencia), valor (tipo dato referencia), fecha de comercio, cantidad, precio unitario, tipo (compra o venta) y estatus (pendiente o completo). La clave primaria de la tabla *Trade* es el número de comercio. Defina la integridad referencial o restricciones de SCOPE para limitar el rango de la referencia cliente y la referencia valor. Tanto el tipo como la tabla comercio no tienen padre.
36. Utilizando Oracle 10g, inserte un objeto en la tabla tipo *CommonStock* para acciones comunes de Microsoft. Para administrar las subtablas, también debe insertar el mismo objeto en las tablas tipo *Stock* y *Security*.

37. Utilizando Oracle 10g, inserte un objeto en la tabla tipo *CommonStock* para acciones comunes de Dell Corporation. Para administrar las subtablas, también debe insertar el mismo objeto en las tablas tipo *Stock* y *Security*.
38. Utilizando Oracle 10g, inserte un objeto en la tabla tipo *CommonStock* para acciones comunes de IBM. Ingrese un valor en la columna precios de cierre (tipo serie de tiempo) mediante la especificación del arreglo de valores, el periodo, el tipo de calendario, la fecha de inicio y la duración.
39. Utilizando Oracle 10g, inserte un objeto en la tabla tipo *Bond* para un bono corporativo IBM.
40. Utilizando Oracle 10g, inserte un objeto en la tabla tipo *Customer*. Use 999999 como el número de cliente, John Smith como el nombre de cliente y Denver como la ciudad.
41. Utilizando Oracle 10g, inserte un objeto en la tabla tipo *Customer*. Use 999998 como el número de cliente, Sue Smith como el nombre de cliente y Boulder como la ciudad.
42. Utilizando Oracle 10g, inserte un objeto en la tabla tipo *Holding*. Conecte el objeto de cartera de inversión con el objeto *Security* de Microsoft y el objeto *Customer* de Sue Smith. Use 200 como el número de participaciones poseídas.
43. Utilizando Oracle 10g, inserte un objeto en la tabla tipo *Holding*. Conecte el objeto de cartera de inversión con el objeto *Secutiry* de IBM y el objeto *Customer* de Sue Smith. Use 100 como el número de participaciones poseídas.
44. Utilizando Oracle 10g, inserte un objeto en la tabla tipo *Trade*. Conecte el objeto comercio con el objeto acción común de IBM y el objeto *Customer* de Sue Smith. Use 100 como la cantidad de participaciones negociadas, “comprar” como el tipo de comercio y otros valores de su elección para las otras columnas.
45. Utilizando Oracle 10g, inserte un objeto en la tabla tipo *Trade*. Conecte el objeto comercio con el objeto acción común de Microsoft y el objeto *Customer* de Sue Smith. Use 200 como la cantidad de participaciones negociadas, “comprar” como el tipo de comercio y otros valores de su elección para las otras columnas.
46. Utilizando Oracle 10g, inserte un objeto en la tabla tipo *Trade*. Conecte el objeto comercio con el objeto bono corporativo de IBM y el objeto *Customer* de John Smith. Use 150 como la cantidad de participaciones negociadas, “comprar” como el tipo de comercio y otros valores de su elección para las otras columnas.
47. Utilizando Oracle 10g, actualice la columna de referencia de cliente del objeto *Holding* del problema 41 al objeto *Customer* de John Smith.
48. Utilizando SQL:2003, actualice la columna de referencia de cliente del objeto *Trade* del problema 44 al objeto *Customer* de John Smith.
49. Utilizando Oracle 10g, escriba un enunciado SELECT para listar los valores poseídos por clientes de Denver. Sólo mencione los valores con más de 100 participaciones poseídas. Incluya en el resultado nombre de cliente, símbolo y participaciones poseídas.
50. Utilizando Oracle 10g, escriba un enunciado SELECT para listar los valores poseídos por clientes de Boulder. Incluya en el resultado nombre de cliente, símbolo de valores, número de comercio, fecha de comercio, cantidad comercializada y precio unitario.
51. Utilizando Oracle 10g, escriba un enunciado SELECT para listar nombre de cliente, símbolo de valores y número de acciones poseídas por cada cliente de Denver.
52. Utilizando Oracle 10g, escriba un enunciado SELECT para listar nombre de cliente, símbolo de valores, número de comercio, fecha de comercio, cantidad comercializada y precio unitario para compras de acciones comunes por clientes de Boulder.

## Referencias para ampliar su estudio

Este capítulo proporcionó una introducción detallada a una materia amplia y profunda. Para más detalles, se le recomienda consultar libros especializados, artículos y sitios web. Las fuentes más definidas acerca de SQL:2003 son los documentos estándar disponibles del *InterNational Committee for Information Technology Standards* ([www.incits.org](http://www.incits.org)). La *Whitemarsh Information Systems Corporation* ([www.wiscorp.com/SQLStandards.html](http://www.wiscorp.com/SQLStandards.html)) mantiene información clave acerca de los estándares de SQL. Como los documentos de estándares son más bien difíciles de leer, es posible que prefiera los libros de SQL:1999 de Gulutzan y Pelzer (1999) y Melton y Simon (2001). Hasta el momento de terminar este capítulo (junio de 2005), no ha aparecido libro alguno acerca de SQL:2003. Los sitios *DBAZine* ([www.dbazine.com](http://www.dbazine.com)) y *DevX Database Zone* ([www.devx.com](http://www.devx.com)) tienen consejos prácticos acerca de bases de datos relacionales de objetos. Para más detalles acerca de las características relacionales de objeto en Oracle 10g, debe consultar la documentación de bases de datos en línea en *Oracle Technology Network* ([www.oracle.com/technology](http://www.oracle.com/technology)).

# Glosario

---

## A

**acciones en filas referenciadas** posibles acciones en respuesta a la eliminación de una fila referenciada o a la actualización de la llave primaria de una fila referenciada. Las posibles acciones son restricción (no permitir la acción en la columna referenciada), cascada (realizar la misma acción en las filas relacionadas), anular (igualar a nulo el valor de la llave foránea de las filas relacionadas) y por omisión (igualar al valor por omisión de la llave foránea de las filas relacionadas). Véase también filas referenciadas.

**acelerar** disminución del tiempo para completar una tarea con capacidad adicional de cómputo en el procesamiento distribuido de bases de datos. La aceleración mide los ahorros en tiempo mientras que conserva la tarea de forma constante. La aceleración se mide mediante la razón del tiempo de finalización con la configuración original entre el tiempo de finalización con la capacidad adicional. Véase también sistema de administración de bases de datos distribuidas y escalabilidad.

**actualización del escolar perdido** variación del problema de la actualización perdida. La palabra *escolar* es una ironía, ya que el problema de actualización del escolar perdido difiere ligeramente del tradicional problema de actualización perdida. La única diferencia esencial entre el problema de actualización del escolar perdido y el problema de actualización perdida es que la transacción A termina antes de que la transacción B modifique los datos que tienen en común. Véase también actualización.

**actualización perdida** problema del control de concurrencias en el que una actualización del usuario sobrescribe otra actualización. Véase también actualización del escolar perdido.

**acumulación de la tabla de hechos** tabla de hechos que registra el estado de varios eventos en lugar de uno solo. Cada columna de ocurrencia de un evento puede representarse por una llave foránea hacia la tabla de tiempo junto con una columna de la hora del día, en caso de ser necesario.

**administración de recursos de información** amplia filosofía de la administración que busca utilizar la tecnología de la información como una herramienta para el procesamiento, distribución e integración de la información de manera acertada y organizada.

**administración del conocimiento** uso de la tecnología de la información junto con las capacidades de procesamiento de información humanas y los procesos de una organización para respaldar una adaptación rápida al cambio.

**administrador de bases de datos (DBA)** posición de apoyo que se especializa en administrar bases de datos individuales. También, el manejador de bases de datos.

**administrador de datos (DA)** posición gerencial que lleva a cabo la planeación y establecimiento de políticas para los recursos de información de una organización en su totalidad.

**alcance de las actualizaciones diferidas** alcance utilizado por un gerente de recuperación para registrar los cambios de una base de datos en disco. En este alcance, las actualizaciones de la base de datos se escriben solamente después de que se completa una transacción. Para recuperar una base de datos, sólo se utiliza la operación rehacer.

**álgebra relacional** conjunto de operadores para manipular las bases de datos relacionales. Cada operador usa una o dos tablas como entrada y genera una nueva tabla como salida.

**algoritmo de enlace** algoritmo para implementar el operador de join. Un componente de optimización de consultas selecciona el algoritmo de enlace menos costoso para cada operación de enlace de una consulta. Los algoritmos de enlace más comunes son los ciclos anidados, ordenar-combinar, enlace híbrido, enlace hash y enlace estrella.

**almacenamiento volátil** almacenamiento que pierde su estado cuando se desconecta de la corriente eléctrica. La memoria principal es típicamente volátil. El almacenamiento no volátil no pierde su estado cuando se desconecta de la corriente. Un disco duro es ejemplo de almacenamiento no volátil.

**analista/programador** profesional de los sistemas de información que es responsable de obtener los requerimientos, diseñar aplicaciones e implementar los sistemas de información. Un analista/programador puede crear y utilizar vistas externas para desarrollar formularios, reportes y otras partes de un sistema de información.

**anomalía de modificación** efecto colateral inesperado que ocurre cuando se modifican datos en una tabla con exceso de redundancias.

**ANSI** Instituto Nacional de Estándares de América, uno de los grupos responsables de los estándares SQL.

**aproximación de actualización inmediata** aproximación utilizada por el administrador de recuperación para grabar modificaciones de la base de datos en el disco. En esta aproximación, la actualización de la base de datos se graba en el disco cuando suceden las modificaciones, pero después de las actualizaciones de las bitácoras. Para restaurar una base de datos puede necesitarse tanto las operaciones de deshacer como de rehacer. Véase también aproximación a una actualización diferida y protocolo para generar la bitácora.

**archivo** conjunto de datos en un dispositivo permanente de almacenamiento, como un disco duro. Los datos o registros físicos del archivo están organizados para apoyar el procesamiento eficiente. Los archivos son parte del esquema interno de una base de datos.

**archivo Btree** estructura de archivos muy popular incluida en la mayoría de los DBMSs porque proporciona un gran desempeño en las búsquedas por llaves y secuenciales. Un archivo Btree es un árbol balanceado de formas múltiples. La variación más popular de un Btree es un B+tree, en el cual se almacenan todas las llaves redundantes en las hojas nodo. Un B+tree proporciona un mejor desempeño en búsquedas secuenciales y por rangos. Un Btree puede utilizarse como una estructura de archivos primaria o secundaria.

**archivo hash** estructura especializada de archivos que soporta búsquedas por llave. Los archivos hash transforman el valor de la llave en una dirección que provee un acceso rápido. Los archivos hash pueden tener un pobre desempeño para accesos secuenciales. Un archivo hash puede ser estático (requiere de reorganización periódica) o dinámico (no requiere de reorganización periódica). Una estructura hash puede usarse como llave primaria o como estructura de archivos secundaria.

**archivo secuencial** organización sencilla de archivos en la que los registros están almacenados en el orden de inserción mediante el valor de una llave. Los archivos secuenciales son fáciles de mantener y proporcionan un buen desempeño para procesar grandes cantidades de registros.

**arquitectura cliente-servidor** conjunto de componentes entre computadoras (clientes y servidores) conectadas mediante una red. La arquitectura cliente-servidor soporta el procesamiento eficiente de mensajes (solicitudes de servicio) entre clientes y servidores.

**arquitectura de abajo-arriba de un data warehouse** arquitectura para un data warehouse en la cual los data marts se construyen de acuerdo con los departamentos de usuarios. Si surge la necesidad para un modelo de datos corporativo, los data marts evolucionarán hacia un data warehouse. Véase también la arquitectura de dos capas y de tres capas de un data warehouse.

**arquitectura de compartir nada (SN)** arquitectura para el procesamiento paralelo de bases de datos en la que cada procesador tiene su propia memoria y discos. En esta arquitectura, los datos deben estar divididos entre los procesadores.

**arquitectura de compartir todo (SE)** arquitectura para el procesamiento paralelo de bases de datos en la que la memoria y los discos se comparten entre un conjunto de procesadores. El enfoque SE generalmente se refiere al multiprocesamiento simétrico de cómputo, más que a la arquitectura paralela de bases de datos.

**arquitectura de data warehouse de tres capas** arquitectura para un data warehouse en la que los departamentos de usuarios acceden a los data marts en lugar de al data warehouse. Un proceso de extracción que incluye la actualización periódica de un data warehouse que actualiza los data marts. Véase también arquitectura de dos capas de un data warehouse y arquitectura de abajo hacia arriba de un data warehouse.

**arquitectura de discos compartidos (SD)** arquitectura para el procesamiento paralelo de bases de datos en la que cada procesador tiene su memoria privada, pero los discos se comparten entre todos los procesadores.

**arquitectura de discos en clúster (CD)** arquitectura para el procesamiento paralelo de bases de datos en la que los procesadores de cada clúster comparten todos los discos, pero no comparten entre clústeres.

**arquitectura de dos capas** arquitectura cliente-servidor en la que un cliente PC y un servidor de bases de datos interactúan de forma directa para solicitar y transferir datos. El cliente PC contiene el código de interfase de usuario, el servidor contiene el acceso lógico a los datos y el cliente PC y el servidor comparten la validación y la lógica de negocio. Véase también arquitectura de tres capas y arquitectura de capas múltiples.

**arquitectura de dos capas de un data warehouse** arquitectura para un data warehouse en la que los departamentos de usuarios usan de forma directa el data warehouse en lugar de data marts más pequeños. Véase también arquitectura de tres capas de un data warehouse y arquitectura de abajo-arriba de un data warehouse.

**arquitectura de los tres esquemas** arquitectura para dividir las descripciones de una base de datos. La arquitectura de los tres esquemas contiene el nivel externo o de usuario, el nivel conceptual y el nivel interno o físico. La arquitectura de los tres esquemas se propuso como una forma de conseguir que los datos fuesen independientes.

**arquitectura de nada en clúster (CN)** arquitectura para el procesamiento paralelo de bases de datos en la que los procesadores de cada clúster no comparten ningún recurso, pero en donde cada clúster puede manipularse para que trabaje en paralelo llevando a cabo cierta tarea.

**arquitectura de objetos grandes** arquitectura para bases de datos de objetos en la que los objetos grandes (binarios o texto) se almacenan en una base de datos junto con el software externo para manipular los objetos grandes.

**arquitectura de servicios web** arquitectura que soporta el comercio electrónico entre las organizaciones. Un conjunto de estándares de Internet relacionados que soportan la alta interoperabilidad entre los solicitantes de servicios, proveedores de servicios y registros de servicios. El estándar más importante es el lenguaje de descripción de servicios web, utilizado por los solicitantes de servicios, proveedores de servicios y registros de servicios.

**arquitectura de tres capas** arquitectura cliente-servidor con tres capas: una PC cliente, un servidor de bases de datos y un middleware o servidor de aplicaciones. Véase también arquitectura de dos capas y arquitectura de capas múltiples.

**arquitectura especializada de servidor de medios** arquitectura para las bases de datos de objetos en donde un servidor dedicado administra los datos complejos por fuera de la base de datos. Los programadores utilizan una interfase de programación de la aplicación para acceder a los datos complejos.

**arquitectura multinivel** arquitectura cliente-servidor con más de tres capas: una PC cliente, un servidor de bases de datos, un servidor de middleware y servidores de aplicaciones. Los servidores de aplicaciones llevan a cabo la lógica del negocio y administran datos de tipo especializado, como las imágenes. Véase también arquitectura de doble nivel y arquitectura de triple nivel.

**ascenso (roll-up)** operador de cubos de datos que soporta la navegación de un nivel más específico de una dimensión hacia un nivel más general de una dimensión. El operador roll-up requiere una dimensión jerárquica. También véase descenso (drill-down).

**aserción** la categoría más general de restricciones de integridad incluida en SQL: 2003. Una aserción puede involucrar una sentencia SELECT de complejidad arbitraria. Para definir aseraciones en SQL:2003 se utiliza la sentencia CREATE ASSERTION.

**atributo** propiedad de entidad o relación. Cada atributo tiene un tipo de datos que define los valores y operaciones permitidas. Atributo es sinónimo de campo y columna.

**autoenlace (self-join)** enlace entre una tabla y ella misma (dos copias de la misma tabla). Generalmente un autoenlace se usa para hacer consultas sobre relaciones autorreferenciadas.

## B

**balanceo de cargas** problema del procesamiento paralelo de bases de datos. El balanceo de cargas involucra la cantidad de trabajo colocada en los distintos procesadores de un clúster. Lo ideal es que cada procesador tenga la misma cantidad de trabajo para aprovechar de forma total el clúster. La arquitectura que no comparte nada es más sensible al balanceo de cargas dada la necesidad del particionamiento de datos. Véase también arquitectura que no comparte nada y arquitectura en clúster que no comparte nada.

**banda** conjunto de registros físicos que pueden leerse o escribirse en paralelo en un almacén RAID. Normalmente, una banda contiene un conjunto de registros físicos adyacentes.

**base de datos operacional** base de datos que soporta las funciones diarias de una organización. Las bases de datos operacionales soportan de forma directa la mayoría de las funciones, tales como el procesamiento de órdenes, fabricación, cuentas por pagar y distribución de productos.

**base de datos** colección de datos persistentes que pueden compartirse e interrelacionarse.

**bitácora de transacciones (log transaction)** tabla que contiene la historia de los cambios de la base de datos. El administrador de recuperaciones usa la bitácora para recuperarse de las fallas.

**búfer** área en la memoria principal que contiene los registros físicos de la base de datos transferidos desde el disco.

## C

**candado** herramienta fundamental del control de concurrencias. Un candado hecho sobre un elemento de una base de datos previene que otras transacciones lleven a cabo acciones conflictivas sobre el mismo elemento. Véase también candado exclusivo, candado dedicado y candado compartido.

**candado compartido** candado que permite que algunos usuarios lean un elemento de la base de datos, pero que evita que dichos usuarios modifiquen el valor del elemento de la base de datos. Los bloqueos compartidos entran en conflicto con los bloqueos exclusivos, pero no con otros bloqueos compartidos. Un bloqueo compartido indica que un usuario

leerá pero no modificará el valor del elemento de una base de datos. También se le conoce como un candado S.

**candado exclusivo** bloqueo que evita que otros usuarios accedan a un elemento de la base de datos. Los candados exclusivos no se acoplan con todos los demás tipos de candado (compartido, otros candados exclusivos e intento). Un candado exclusivo indica que un usuario cambiará el valor de un elemento de la base de datos. También se le conoce como candado X.

**cardinalidad** restricción sobre el número de entidades que participan en una relación. En un diagrama de entidad-relación, el número mínimo y máximo de entidades se especifican en ambos lados de la relación.

**ciclo de interrelaciones** conjunto de interrelaciones ordenadas en un ciclo que comienza y termina con el mismo tipo de entidad. Debe examinar los ciclos de relaciones para determinar el momento en que se puede deducir una relación a partir de otras relaciones.

**ciclo de vida de la información** los estados de transformación de la información en una organización. Los estados típicos del ciclo de vida de la información incluyen la adquisición, almacenaje, protección, formateo, diseminación y uso.

**ciclos anidados** algoritmo de enlace que usa una tabla externa y una tabla interna. El algoritmo de ciclos anidados es el algoritmo de enlace más general que puede usarse para evaluar todas las operaciones de enlace, no sólo los enlaces de igualdad. El algoritmo de ciclos anidados se desempeña bastante bien cuando existen pocas filas en la tabla externa o cuando todas las páginas de la tabla interna caben en la memoria. Un índice sobre la llave foránea de una columna de enlace permite el uso eficiente del algoritmo de ciclos anidados cuando existen condiciones restrictivas en la tabla padre. Véase también enlace hash y combinar orden.

**clase** colección de objetos. La definición de una clase incluye definiciones de variables para los datos de objetos y métodos para los procedimientos de los objetos.

**cliente** programa que envía solicitudes a un servidor para acceder o actualizar una base de datos.

**clúster** acoplamiento entre dos o más computadoras para que se comporten como una sola. Los clústers proporcionan flexibilidad adicional para el procesamiento paralelo de bases de datos. Véase también arquitectura de discos en clúster y arquitectura de nada en clúster.

**clúster de aplicación real de Oracle (RAC)** tecnología de Oracle para el procesamiento de bases de datos paralelas. Oracle RAC usa la arquitectura de discos en clúster.

**coherencia de caché** problema del procesamiento de bases de datos paralelo que utiliza arquitecturas de discos compartidos. La coherencia de caché involucra la sincronización entre las memorias locales y el almacenamiento de disco compartido. Después de que un procesador accede a una página en disco, la imagen de esta página permanece en el caché asociado con el procesador en cuestión. Si otro procesador modificó la página en su propio búfer, ocurre una inconsistencia. Véase también arquitectura compartida de discos y arquitectura agrupada de discos.

**coincidencia exacta de cadenas de caracteres** búsqueda de una cadena de caracteres usando el operador de igualdad. Véase también coincidencia inexacta de una cadena de valores.

**coincidencia inexacta de una cadena de valores** búsqueda por un patrón de semejanzas en lugar de por una cadena de valores. En SQL, la coincidencia inexacta de una cadena de valores usa el operador LIKE y los caracteres del patrón de semejanzas.

**colisión** condición que involucra inserciones en un archivo hash. Una colisión ocurre cuando dos o más registros se insertan en la misma ubicación. Una función hash transforma el valor de la llave en una dirección para cada registro. Véase también archivo hash.

**columna** campo o atributo de tabla. Cada columna tiene un tipo de datos que define valores y operaciones permisibles. Columna es sinónimo de campo y atributo.

**compartido** característica fundamental de las bases de datos. Compartir significa que una base de datos puede tener múltiples usos y usuarios. Una base de datos grande puede tener cientos de funciones que la utilicen, así como miles de usuarios que accedan a ella de forma simultánea.

**compatibilidad de unión** requerimiento de los operadores de unión, intersección y diferenciación del álgebra relacional. La compatibilidad de unión requiere que ambas tablas tengan el mismo número de columnas y que cada columna correspondiente tenga un tipo de datos compatible.

**condición de fila** comparación que no incluye una función agregada. Las condiciones de las filas se evalúan en la cláusula WHERE.

**condición grupal** comparación que incluye y agrega funciones, tales como SUM o COUNT. Las condiciones grupales no pueden evaluarse sino hasta después de que se haya evaluado la cláusula GROUP BY.

**conjunto de datos modificados** datos obtenidos de un sistema origen para actualizar un data warehouse. El conjunto de datos modificados incluye extracciones periódicas de los datos fuente. Para deducir los datos modificados, una operación de diferenciación usa los dos conjuntos de datos más recientes. Véase también datos modificados cooperativos, datos modificados almacenados en bitácora y datos modificados consultables.

**consejo de procesamiento de transacciones (TPC)** organización que desarrolla estándares y comparaciones de dominios específicos, y que publica los resultados. El TPC ha desarrollado mediciones para el procesamiento de transacciones de captura de órdenes, decisiones a la medida que soporten consultas, decisiones de soporte a reportes de negocios y procesamiento de transacciones de comercio electrónico. Véase también medición.

**consulta (query)** solicitud para extraer datos útiles. La formulación de consultas involucra traducir un problema a un lenguaje que el DBMS entienda (como la sentencia SELECT de SQL).

**consulta anidada** consulta dentro de una consulta. En una sentencia SQL SELECT, la sentencia SELECT puede ser parte de las condiciones de las cláusulas WHERE y HAVING. Véase las consultas anidadas tipo I y tipo II para revisar las dos

variantes. También se le conoce como subconsulta o consulta interna.

**consulta anidada tipo I** consulta anidada en la que la consulta interna no hace referencia a ninguna tabla usada en la consulta anidada externa. Las consultas anidadas tipo I pueden usarse cuando existen problemas para algunos enlaces y problemas de diferenciación.

**consulta anidada tipo II** consulta anidada en la que la consulta interna hace referencia a alguna tabla usada en la consulta anidada externa. Las consultas anidadas tipo I pueden usarse cuando existen problemas de diferenciación, pero deben evitarse para problemas de enlaces.

**consulta de actualización de uno-a-muchos (1-M)** tipo de vista actualizable de Microsoft Access que involucra una o más relaciones 1-M.

**control de accesos discrecional** forma más común de controlar la seguridad soportada por la mayoría de los DBMS comerciales, en el momento de asignar a los usuarios derechos o privilegios sobre partes específicas de la base de datos.

**control de accesos obligatorio** alcance de la seguridad de bases de datos para bases de datos altamente sensibles y estáticas. En los alcances de los controles obligatorios, a cada objeto se le asigna un nivel de clasificación y a cada usuario un nivel de permisos. Un usuario puede acceder a un elemento de una base de datos si el nivel de permisos del usuario le proporciona acceso al nivel de clasificación del elemento.

**control incremental** alternativa al control tradicional caché-consistente que incluye un encabezado más reducido, pero que requiere más trabajo de reinicialización. En un control incremental, ninguna página de la base de datos se graba en el disco. En su lugar, las páginas sucias de la base de datos se graban en un orden ascendente según su antigüedad. En el control de tiempo, la posición de la bitácora de la página de datos sucia más antigua se graba para proporcionar un punto de inicio para la restauración. La cantidad de trabajo de restauración puede controlarse mediante la frecuencia de escritura de las páginas sucias. Véase también control.

**control optimista de concurrencias** enfoque del control de concurrencias en el que a las transacciones se les permite acceder a una base de datos sin tener bloqueos. En su lugar, el administrador del control de concurrencias revisa el momento en que ocurre algún conflicto. La revisión puede hacerse antes de que una transacción se termine o después de cada lectura y escritura. El administrador del control de concurrencias puede determinar si ha ocurrido algún conflicto al revisar el tiempo relativo de lecturas y escrituras. Si ocurre algún conflicto, el administrador del control de concurrencias genera un rollback y reinicia la transacción en cuestión.

**cuarta forma normal (4NF)** característica de una tabla. Una tabla se encuentra en 4NF si no contiene ningún MVD no trivial. Un MVD no trivial es un MVD que tampoco es un FD.

**cubo de datos** formato multidimensional en el cual las celdas contienen datos numéricos, llamados medidas, organizados por temas, llamados dimensiones. A un cubo de datos a veces también se le conoce como hipercubo, ya que conceptualmente puede tener un número ilimitado de dimensiones.

**cuerpo de la tabla** sinónimo para las filas de una tabla.

**cursor** construcción del lenguaje de programación de una base de datos que permite el almacenamiento e iteración a través del conjunto de registros devueltos por una sentencia SELECT. Un cursor es semejante a un arreglo dinámico en el que el tamaño del arreglo se determina por el tamaño del resultado de la consulta. Un lenguaje de programación de bases de datos provee sentencias y procedimientos para declarar cursos, abrir y cerrar cursos, posicionarse en los cursos y obtener valores de los cursos.

**cursor implícito de PL/SQL** cursor que puede utilizarse en procedimientos codificados en PL/SQL, el lenguaje de programación de base de datos de Oracle. Un cursor implícito no está explícitamente declarado ni abierto. Es una versión especial de la cláusula de declaración de un FOR, abre, interactúa y cierra una cláusula llamada SELECT de forma local. Un cursor implícito no puede referenciarse desde afuera de la cláusula FOR en la cual está declarado. Véase también cursor y cursos explícitos de PL/SQL.

**cursor PL/SQL explícito** cursor que puede usarse en procedimientos escritos en PL/SQL, el lenguaje de programación de la base de datos Oracle. Un cursor explícito se declara con la sentencia CURSOR en la sección DECLARE. Los cursos explícitos por lo general se manipulan con las sentencias OPEN, CLOSE y FETCH. Los cursos explícitos pueden referenciarse en cualquier lugar dentro de la sección BEGIN. Véase también cursor PL/SQL implícito.

## D

**dado** operador de un cubo de datos en el cual una dimensión se reemplaza por un subconjunto de sus valores. Véase también rebanada.

**data mart** subconjunto o vista de un data warehouse, comúnmente a un nivel departamental o funcional, que contiene todos los datos requeridos para apoyar la toma de decisiones de dicho departamento. Adicionalmente, un data mart aísla a los usuarios departamentales de los datos utilizados por otros departamentos. En algunas organizaciones, un data mart es un data warehouse pequeño, en lugar de la vista del data warehouse.

**data warehouse** repositorio central de datos resumidos e integrados a partir de bases de datos operativas y de fuentes de datos externas.

**datos de cambio cooperativo** datos obtenidos de un sistema origen para actualizar un data warehouse. Los datos de cambio cooperativo incluyen la notificación del sistema de origen usualmente al terminar una transacción con el uso de un disparador. Véase también datos modificados registrados en bitácora y cambio de datos de consulta.

**datos de modificación de registros** datos obtenidos de un sistema origen para refrescar un data warehouse. Los datos de modificación de registro incluyen archivos que registran las modificaciones de los registros u otra actividad del usuario, como bitácoras web o bitácoras de transacciones. Véase también datos de modificación cooperativos, datos de modificación estáticos y datos de modificación para consultas.

**datos modificados** datos de un sistema origen que proveen la base para actualizar un data warehouse. Los datos modificados insertan nuevos datos fuente (inserciones) y modificaciones

a los datos originales (actualizaciones y eliminaciones). Posteriormente, los datos modificados pueden afectar las tablas de hechos y/o de dimensiones. Véase también cambio de datos cooperativos, cambiar datos modificados registrados en la bitácora, fotografía de los datos modificados y datos modificados consultables.

**datos modificados para consulta** datos obtenidos de un sistema origen para refrescar un almacén de datos. Como los datos modificados para consulta provienen de forma directa de una fuente de datos a través de una consulta, la fuente de datos debe tener marcas de tiempo. Véase también datos modificados cooperativos, datos modificados registrados y datos modificados para un momento dado.

**DBMS corporativo** DBMS que soporta las bases de datos que generalmente son críticas para el funcionamiento de una organización. Los DBMS corporativos por lo general se ejecutan en servidores poderosos y tienen un costo alto. Véase también DBMS de escritorio y DBMS embebido.

**DBMS de escritorio** DBMS que soporta las bases de datos utilizadas por equipos de trabajo y negocios pequeños. Los DBMS de escritorio están diseñados para ejecutarse en computadoras personales y servidores pequeños. Véase también DBMS corporativo y DBMS embebido.

**DBMS distribuido** conjunto de componentes que apoya las consultas de datos residentes en múltiples ubicaciones. Un DBMS distribuido encuentra datos remotos, optimiza consultas globales y coordina transacciones en múltiples ubicaciones. También se conoce como sistema de administración de bases de datos distribuidas (DDBMS).

**DBMS embebido** DBMS que reside en un sistema más grande, incluso en una aplicación o dispositivo, como un Asistente Digital Personal (PDA) o una tarjeta inteligente. Los DBMS embebidos proveen funciones limitadas de procesamiento transaccional, pero demandan pocos requerimientos de memoria, procesamiento y almacenamiento. Véase también DBMS de escritorio y DBMS corporativo.

**DBMS objeto-relacional** DBMS relacional ampliado con un procesador de consultas de objetos para tipos de datos definidos por el usuario. SQL:2003 proporciona el estándar para los DBMS objeto-relacionales.

**DBMS orientado a objetos** nuevo tipo de DBMS diseñado especialmente para objetos. Los DBMS orientados a objetos tienen un procesador de consultas de objetos y un núcleo de objetos. El Grupo de Administración de Datos de Objetos proporciona el estándar para los DBMS orientados a objetos.

**DBMS relacional** sistema que usa el Modelo de datos relacional para administrar un conjunto de datos.

**dependencia de la existencia** entidad que no puede existir a menos que exista otra entidad relacionada. Una relación obligatoria produce una dependencia de la existencia. Véase también relación obligatoria.

**dependencia de múltiples valores (MVD)** restricción que involucra tres columnas. La MVD A →→ B | C (se lee A multidetermina a B o C) significa que 1) un valor dado de A se asocia con una colección de valores B y C; y 2) B y C son relaciones independientes entre A y B, y A y C. Todas las FD también son MVD, pero no todas las MVD son FD.

Una MVD es no trivial si tampoco es una FD. Véase también independencia de relaciones y dependencia funcional.

**dependencia funcional (FD)** restricción con respecto a dos o más columnas de una tabla.  $X$  define  $Y(X \rightarrow Y)$  si existe al menos un valor de  $Y$  para cada valor de  $X$ . Una dependencia funcional es semejante a una restricción de una llave candidata, ya que  $X$  es una llave candidata si  $X$  y  $Y$  se colocan en tablas distintas.

**dependencia transitoria** dependencia funcional derivada por la ley de transitoriedad. Los FDs transitorios no deben registrarse como un insumo del proceso de normalización. Véase también dependencia funcional y ley de transitoriedad.

**dependencias sin comprometer** problema del control de concurrencias en el que una transacción lee datos escritos por otra transacción antes de que la otra transacción finalice. Si la segunda transacción se aborta, la primera transacción leyó datos fantasma que no existen. También se le conoce como lectura sucia.

**descenso (drill-down)** operador de un cubo de datos que soporta la navegación desde un nivel general de una dimensión hasta un nivel más específico de una dimensión. Véase también ascenso (roll-up).

**desnormalización** combinación de tablas para que las consultas sean más sencillas. Desnormalización es lo opuesto a normalización. La desnormalización puede ser útil para mejorar el desempeño de las consultas o para ignorar una dependencia que no ocasione anomalías de almacenamiento significativas.

**determinante** cada columna que aparece a la izquierda de una dependencia funcional. También se conoce como LHS o lado izquierdo.

**diccionario de datos** base de datos especial que describe bases de datos individuales y el entorno de las bases de datos. El diccionario de datos contiene descriptores llamados metadatos que definen el origen, uso, valor y significado de los datos. Véase también metadatos.

**diccionario de recursos de información (IRD)** base de datos de metadatos que describe la información completa del ciclo de vida de la información. El sistema del diccionario de recursos de información administra los accesos a un IRD. También se le conoce como repositorio.

**diferencia** operador del álgebra relacional que combina filas de dos tablas. El operador de diferencia extrae únicamente las filas que pertenecen a la primera tabla. Las dos tablas deben ser compatibles con el operador unión para poder usar el operador de diferencia.

**dimensión** etiqueta o tema que organiza los datos numéricos en un cubo de datos. Una dimensión contiene valores conocidos como miembros, tales como la ubicación de la dimensión que tenga miembros como países. Las dimensiones pueden organizarse en jerarquías, formadas por niveles de apoyo a las operaciones del cubo de datos para particularizar o generalizar. Una jerarquía de dimensiones puede hacerse mostrando las relaciones entre los miembros del mismo nivel de dimensión.

**disparador (trigger)** regla almacenada y ejecutada por un DBMS. Debido a que un disparador incluye un evento, una

condición y una secuencia de acciones, también se le conoce como regla de evento-condición-acción. Los disparadores no eran parte de SQL-92, aunque muchos fabricantes proporcionaban ampliaciones para usarlos. Los disparadores son parte de SQL:2003. Véase también disparadores traslapados.

**disparadores traslapados** dos o más disparadores con el mismo tiempo, granularidad y tabla a la que se aplican. Los disparadores se traslanan cuando una sentencia SQL ocasiona que se ejecuten ambos disparadores. Usted no debe depender de un orden específico de ejecución para los disparadores traslapados. Véase también disparador y procedimiento de ejecución de disparadores.

**disperso** ampliación de celdas vacías en un cubo de datos. Si un gran número de celdas están vacías, el cubo de datos puede desperdiciar mucho espacio y ser lento al procesar. Pueden usarse técnicas especiales de compresión para reducir el tamaño de los cubos de datos dispersos. La dispersión puede ser un problema si se relacionan dos o más dimensiones, como los productos y las regiones donde se venden los productos. Véase también cubo de datos.

**división** operador del álgebra relacional que combina filas de dos tablas. El operador división produce una tabla en la que los valores de una columna de una de las tablas fuente se asocia con todos los valores de una columna de la segunda tabla.

## E

**encabezado de la tabla** conjunto formado por el nombre de la tabla, los nombres de las columnas y el tipo de datos de cada columna.

**encapsulación** principio del cómputo orientado a objetos en el que sólo puede accederse a un objeto a través de su interfase. No pueden utilizarse los detalles internos (variables e implementación de métodos). La encapsulación soporta bajo costo de mantenimiento de software.

**encriptación** acción que involucra la codificación de datos para ocultar su significado. Un algoritmo de encriptación modifica los datos originales (conocidos como texto plano). Para descifrar los datos, el usuario proporciona la llave de encriptación para restaurar los datos encriptados (conocidos como texto cifrado) a su formato original (texto plano).

**enlace (join)** operador del álgebra relacional usado para combinar las filas de dos tablas. El operador de enlace genera una tabla que contiene filas que cumplen una condición que involucra una columna de cada una de las tablas fuente. Véase también equienlace (equi-join) y enlace natural (natural join).

**enlace completo exterior (full outer join)** enlace exterior que genera las filas coincidentes de una parte de la igualdad, así como las filas que no coinciden en ambas tablas.

**enlace de base de datos** concepto clave para las bases de datos distribuidas por Oracle. Un enlace de bases de datos proporciona una conexión unidireccional de una base de datos local a una base de datos remota. Los enlaces de bases de datos permiten que un usuario acceda a los objetos de otro usuario de una base de datos remota sin necesidad de tener una cuenta en el sitio remoto. Cuando se usa un enlace de base de datos, el usuario remoto está limitado por el conjunto de privilegios del usuario propietario.

**enlace de igualdad** operador de enlace donde la condición del enlace involucra la igualdad. Véase también enlace y enlace natural.

**enlace** asociación de un plan de accesos con una sentencia SQL, en la optimización de consultas. En la computación orientada a objetos, un enlace se refiere a la asociación del nombre de un método con su implementación. El enlace puede ser estático (se realiza durante la compilación) o dinámico (se realiza durante la ejecución). El enlace estático es más eficiente pero muchas veces menos flexible que el enlace dinámico. Véase también plan de accesos y mensajes.

**enlace estrella** algoritmo de enlace que combina dos o más tablas en el que existe una tabla hijo relacionada con muchas tablas padre mediante relaciones 1-M. Cada tabla padre debe tener un índice de enlace bitmap. El enlace estrella es el mejor algoritmo de enlace cuando existen muchas condiciones de selección en las tablas padre. El algoritmo de enlace estrella se usa ampliamente para optimizar las consultas sobre los almacenes de datos. Véase también índice bitmap.

**enlace externo (outer join)** operador del álgebra relacional que combina dos tablas. En un enlace externo, las filas que coinciden y las que no coinciden se conservan en el resultado. Véase enlaces externos completos y de un solo lado para revisar las dos variantes de este operador.

**enlace externo de un solo lado (one-sided outer join)** enlace externo que produce las filas que coinciden (la parte del enlace) y las que no coinciden de una sola de las tablas, la tabla designada como origen.

**enlace hash** algoritmo de enlace que usa un archivo hash interno para cada tabla. El algoritmo de enlace hash puede usarse sólo para enlace de igualdad. Los enlaces hash llevan a cabo un mejor ordenamiento cuando las tablas no están ordenadas o no existen índices. Véase también combinación ordenada.

**enlace híbrido** enlace algorítmico que combina los algoritmos de orden-mixto y de ciclos anidados. La tabla externa debe estar ordenada o tener una columna de enlace indexada. La tabla interna debe tener un índice en la columna de enlace. Este algoritmo sólo puede utilizarse para enlaces equitativos. El enlace híbrido tiene un mejor desempeño que el de orden mixto cuando una tabla interna tiene un índice no agrupado en la columna del enlace. Véase también ciclos y tipos de unión.

**enlace natural (natural join)** variación del operador de enlace del álgebra relacional. En un enlace natural, la condición de coincidencia es de igualdad (enlace de igualdad –equijoin–), una de las columnas de coincidencia se elimina de la tabla resultante y las columnas de enlace tienen los mismos nombres sin estar limitadas. Véase también enlace de igualdad y enlace.

**entidad** conjunto de datos, por lo general de cierto tópico, al cual se puede acceder de forma conjunta. Una entidad puede representar una persona, lugar, evento o cosa.

**entidad débil** tipo de entidad que pide prestada una parte o toda su llave primaria a otro tipo de entidad. Una entidad débil también depende de la existencia de otras entidades.

Véase también dependencias identificables y relaciones identificables.

**equivalencia de relaciones** regla sobre la equivalencia entre las relaciones 1-M y M-N. Una relación M-N puede reemplazarse por un tipo de entidad asociativa y dos relaciones 1-M identificables. Véase también tipo de entidad asociativa y relación identificable.

**escalabilidad** cantidad de trabajo que puede completarse mediante el incremento de la capacidad de cómputo, en el procesamiento de bases de datos distribuidas. La escalabilidad mide el tamaño del aumento de trabajo que puede realizarse mientras se conserva el tiempo como constante.

**especificación de nombres** acción por la que se antepone al nombre de una columna el nombre de su tabla. El nombre de la columna por sí solo es una abreviación. Si en una sentencia SQL aparece dos veces el nombre de una columna, el nombre de la columna debe estar limitado por el nombre de su tabla. La combinación del nombre de tabla y de columna debe ser única entre todas las tablas de la base de datos.

**esquema** definición de las partes conceptuales, externas e internas de una base de datos. En el nivel conceptual, un esquema es un diagrama que ilustra las entidades y relaciones de una base de datos. Véase también arquitectura de los tres esquemas.

**esquema conceptual** descripción de datos que abarca todas las entidades y relaciones de una base de datos. Al esquema conceptual le preocupa el significado de la base de datos, no su implementación física. Véase también esquema, esquema interno, vista externa y arquitectura de triple esquema.

**esquema de constelación** representación del modelado de datos para bases de datos multidimensionales. En una base de datos relacional, el esquema de constelación contiene varias tablas de hechos en el centro que se relacionan con tablas de dimensiones. Generalmente las tablas de hechos comparten algunas tablas de dimensiones. Véase también esquema de copos de nieve, esquema estrella, tabla de hechos y tabla de dimensiones.

**esquema de copo de nieve** representación de un modelo de datos para bases de datos multidimensionales. En una base de datos relacional, un esquema de copo de nieve tiene varios niveles o tablas de dimensión relacionadas a una o más tablas de hechos. Debe usar el esquema del copo de nieve en lugar del esquema de estrella para tablas de dimensiones pequeñas que no se encuentren en la tercera forma normal. Véase también esquema de estrella, esquema de constelación, tabla de hechos y tabla de dimensiones.

**esquema estrella** representación de un modelo de datos para bases de datos multidimensionales. En las bases de datos relacionales, un esquema de estrella tiene una tabla de hechos en el centro relacionada con tablas de dimensión. Véase también esquema de copos de nieve, esquema de constelación, tabla de hechos y tabla de dimensiones.

**esquema interno** descripción física de una implementación de base de datos. Véase también esquema, esquema conceptual, vista externa y arquitectura de esquema de árbol.

**estilo de producto cruz** modo de formular enlaces (joins) en una sentencia SELECT. El estilo de producto cruz enumera las tablas de la cláusula FROM y las condiciones de enlaces de la cláusula WHERE.

**estilo del operador de enlace** manera de formular enlaces en una sentencia SELECT. El estilo del operador de enlace lista las operaciones de enlace en la cláusula FROM usando las palabras clave INNER JOIN y ON.

**estrategia de integración** mezcla de los enfoques incrementales y paralelos para integrar un conjunto de vistas. Las vistas son divididas en subtipos. La integración incremental se usa para cada subtipo de vistas. La integración paralela es comúnmente aplicada a los ERDs resultantes de la integración de los subtipos de vistas.

**estructura de archivos secundaria** estructura de archivos que almacena datos llave junto con punteros hacia datos que no son llave. Los índices de tipo bitmap pueden ser únicamente estructuras de archivos secundarias. Las estructuras hash y Btrees pueden ser estructuras de archivos primarias o secundarias. Véase también estructura primaria de archivo.

**estructura del formulario** jerarquía que ilustra la relación entre los campos del formulario. A un grupo de campos del formulario se le conoce como nodo. La mayoría de los formularios tienen una estructura simple con un nodo padre (formulario principal) y un nodo hijo (subformulario).

**estructura primaria de archivos** estructura de archivos que almacena tanto los datos clave como los que no lo son. Los archivos secuenciales sólo pueden ser estructuras de archivos. Las estructuras hash y Btrees pueden ser estructuras de archivos primarias o secundarias. Véase también estructura secundaria de archivos.

**expresión** combinación de constantes, nombres de columnas, funciones y operadores que producen un valor. Las expresiones pueden usarse en cualquier lugar donde aparezca el nombre de la columna en ciertas condiciones y columnas resultantes.

**expresión lógica** expresión que genera un valor (booleano) verdadero o falso. Las expresiones lógicas pueden incluir comparaciones y operadores lógicos (AND, OR, NOT, etcétera).

## F

**fila referenciada** fila de una tabla padre que tiene un valor de llave primaria idéntico a los valores de llaves foráneas de las filas de una tabla hijo. Véase también acciones sobre las filas referenciadas.

**fila sin coincidencia** fila que no se combina con una fila de una segunda tabla con el fin de satisfacer la condición de un enlace. La fila no estará en el resultado de la operación de enlace, pero estará en el resultado de una operación de enlace externo.

**flujo de trabajo** colección de tareas relacionadas estructuradas para cumplir un proceso de negocio.

**forma natural** regla sobre las dependencias permitidas.

forma normal Boyce-Codd (BCNF) característica de una tabla. Una tabla se encuentra en BCNF si cada uno de sus determinantes es una llave candidata. BCNF es una definición revisada de 3NF.

**formulario** documento usado en un proceso de negocio, formateado para proporcionar un modo adecuado de capturar y editar los datos. Un formulario se diseña para apoyar una

tarea del negocio, como procesar una orden, registrar clases o realizar una reservación en una aerolínea.

**formulario de tipo de entidad** formulario que se obtiene a partir de la llave primaria, en el proceso de análisis del formulario. El formulario de tipo de entidad debe colocarse en el centro del ERD.

**formulario jerárquico** ventana con formato para la captura y despliegue de datos usando una parte fija (formulario principal) y otra variable (subformulario). En el formulario principal puede mostrarse un registro y muchos registros relacionados en el subformulario.

**formulario principal** parte fija de un formulario jerárquico. El formulario principal muestra un registro a la vez.

**forzar la escritura** habilidad de controlar el momento en el que los datos se transfieren a un almacenamiento no volátil. Esta característica es fundamental para la administración de recuperación de datos. El evento de forzar la escritura comúnmente ocurre al final de la transacción y en el momento del control.

**fragmento** subconjunto de una tabla que reside en sitios. Los fragmentos pueden ser subconjuntos horizontales (operador de restricción), subconjuntos verticales (operador proyecto), subconjuntos horizontales derivados (operador semi-join) y combinaciones de estos tipos de fragmentos. Véase también operador de semi-join, transparencia de la fragmentación, transparencia de la ubicación y transparencia del mapeo local.

**franjas** técnica para colocar los registros físicos en un almacen RAID para que sean posibles las operaciones de lectura o escritura en paralelo.

**frontera de la transacción** importante decisión del diseño de una transacción en la que una aplicación formada por una colección de sentencias SQL se divide en una o más transacciones. Las decisiones sobre la frontera de una transacción pueden afectar (de manera positiva o negativa) a la salida de la transacción.

**función agregada** función para resumir o hacer estadísticas. Las funciones agregadas estándar en SQL son MIN, MAX, COUNT, SUM y AVG.

**función de partición de una base de datos (DPF)** tecnología de IBM para el procesamiento paralelo de bases de datos. La opción DPF de la edición corporativa de IBM DB2 utiliza la arquitectura de clúster nada.

## G

**granularidad del candado** tamaño del elemento bloqueado de la base de datos. La granularidad del candado es un intercambio de ventajas y desventajas entre el tiempo de espera (cantidad de concurrencias permitidas) y la sobrecarga (número de bloqueos que se mantienen).

## H

**herencia** característica del modelo de datos que soporta compartir los atributos entre un supertipo y un subtipo. El subtipo hereda atributos desde su supertipo. En SQL:2003, la herencia aplica tanto a tipos definidos por el usuario como a las familias de subtablas. La herencia permite compartir código y datos entre objetos similares.

**herramienta CASE** herramienta que facilita el desarrollo de sistemas de información y bases de datos. Las herramientas CASE incluyen funciones para dibujar, analizar, crear prototipos y diccionarios de datos. CASE es un acrónimo de Ingeniería de Software Asistida por Computadora.

**herramientas ETL** herramientas de software para la extracción, transformación y carga de datos modificados a partir de las fuentes de datos a un almacén de datos. Las herramientas ETL eliminan la necesidad de escribir código personalizado para muchas de las tareas de mantenimiento de un data warehouse.

**histograma** gráfica bidimensional donde el eje X representa el rango de columnas y el eje Y representa el número de filas que contiene el rango de valores. Los histogramas incluyen una distribución más detallada de datos que el supuesto de valores uniformes. Los histogramas son parte del perfil de una tabla. Un histograma tradicional o de amplitud equitativa tiene el mismo valor para el ancho de las columnas pero un número variable de filas en cada barra. Un histograma de altura equitativa tiene un rango de columnas con tamaños variables pero aproximadamente el mismo número de filas en cada barra. La mayoría de los DBMSs usan histogramas de altura equitativa, ya que el máximo error estimado esperado puede controlarse de forma sencilla incrementando el número de rangos.

**hoja de datos** manera de desplegar una tabla en la que los nombres de las columnas aparecen en la primera fila y el cuerpo en el resto de las columnas. El término hoja de datos es propio de Microsoft Access.

**HOLAP** acrónimo para Procesador Analítico Híbrido en Línea. HOLAP es una implementación que combina la ingeniería de almacenamiento MOLAP y ROLAP. HOLAP incluye almacenamiento de datos relacional y multidimensional, así como la combinación de ambas fuentes (relacional y multidimensional) para operaciones de datos de un cubo. Véase también MOLAP y ROLAP.

**homónimo** grupo de palabras en una vista integrada que tienen el mismo sonido e incluso la mismas letras, pero distintos significados. Los homónimos se originan por el contexto de su uso. Véase también sinónimo.

## I

**identificador de dependencia** elemento que incluye una entidad débil y una o más relaciones identificadoras. Véase también entidad débil y relación identificadora.

**independencia de datos** base de datos que debe tener una identidad separada de las aplicaciones que la utilizan (programas de cómputo, formularios y reportes). La identidad separada permite que la definición de la base de datos cambie sin afectar a las aplicaciones relacionadas.

**independencia de las relaciones** relación que puede deducirse de dos relaciones independientes.

**índice** estructura de archivos secundaria que provee una ruta alternativa hacia los datos. Los índices comúnmente contienen sólo las llaves de los valores, no los otros campos de un registro lógico. Los índices pueden organizarse como Btrees, estructuras hash o estructuras de mapas de bits. Véase también archivos Btree, archivos hash, índices de mapas de bits.

**índice de bitmaps** estructura de archivos secundaria formada por una columna con un valor y un bitmap. Un bitmap contiene una posición de un bit por cada fila de una tabla referenciada. Un índice de una columna bitmap hace referencia a las filas que contienen la columna con el valor. Un índice que une un bitmap hace referencia a las filas de una tabla hijo que se une con las filas de una tabla padre que contiene la columna. Los índices de bitmaps funcionan bien para columnas estables con pocos valores típicos de tablas en un almacén de datos. Véase también unión tipo estrella.

**índice desagrupado** índice en el que el orden de los registros de los datos no está relacionado con el orden del índice. Un índice desagrupado siempre es una estructura de archivos secundaria. Véase también selección de índices y estructura secundaria de archivos.

**índice en clúster** índice en el que el orden de los registros de datos se acerca al orden del índice. Un índice en clúster puede organizarse como un índice primario o como una estructura secundaria de archivos. Véase también selección de índices, índices que no están en clúster, estructura de archivos primarios y estructura de archivos secundarios.

**ingeniería en reversa** habilidad para extraer definiciones de un sistema de administración de bases de datos origen y usarlas para crear un ERD y las propiedades de un diccionario de datos. Véase también herramienta CASE e ingeniería de avanzada.

**ingeniería hacia adelante** habilidad para generar definiciones para un sistema de administración de bases de datos deseado a partir de un ERD y de las propiedades de un diccionario de datos. Véase también herramientas CASE e ingeniería en reversa.

**integración incremental** enfoque de la vista de integración donde un ERD parcial se une con la siguiente vista. Para integrar  $n$  vistas, existen  $n - 1$  pasos de integración.

**integración paralela** enfoque para revisar la integración en donde todas las vistas se integran en un solo paso. Para integrar  $n$  vistas, existen  $n$  pasos de diseño de vistas y un paso de integración. Los pasos del diseño de vistas pueden realizarse en paralelo por distintos equipos de diseñadores.

**integridad de la entidad** restricción que involucra llaves primarias. No existen dos filas de una tabla que contengan el mismo valor para la llave primaria. Adicionalmente, ninguna fila puede contener un valor nulo para cualquier columna de una llave primaria.

**integridad referencial** restricción de integridad que involucra una llave candidata de una tabla con una llave foránea de otra tabla. Sólo pueden almacenarse dos tipos de valores en una llave foránea: 1) un valor que coincide con el valor de una llave candidata en alguna fila de una tabla que contiene la llave candidata asociada, o 2) un valor nulo. Véase también llave primaria, llave candidata y llave foránea.

**intento de candado** candado en un objeto grande de la base de datos (como una tabla) que muestra la intención de bloquear un objeto menor contenido en el objeto mayor. Los intentos de candado aligeran el bloqueo cuando se bloquea una gran cantidad de objetos y permite una eficiente detección de conflictos entre bloqueos de objetos de variación granular.

**interface de llamada-nivel (CLI)** tipo de lenguaje para integrar el lenguaje de programación con un lenguaje no procedural, tal como SQL. Un CLI incluye un conjunto de procedimientos y un conjunto de definiciones de tipos de datos para manipular los resultados de las sentencias SQL. Los CLIs más usados, Conectividad Abierta de Base de datos (ODBC) apoyada por Microsoft y la Conectividad de Base de datos a través de Java (JDBC) apoyada por Oracle, son compatibles con SQL:2003 CLI.

**interfase de lenguaje procedural** método para combinar un lenguaje no procedural de tipo SQL con un lenguaje de programación tipo COBOL o Visual Basic. El SQL embebido es un ejemplo de una interfase de lenguaje procedural.

**interfase del nivel de sentencias** estilo de lenguaje para integrar un lenguaje de programación con un lenguaje no procedural, como SQL. Una interfase de nivel de sentencias incluye modificaciones a la sintaxis del lenguaje de programación para colocar las sentencias de SQL embebido. SQL:2003 especifica las sentencias para establecer las conexiones a las bases de datos, ejecutar sentencias SQL, usar los resultados de una sentencia SQL, asociar las variables de los programas con las columnas de la base de datos, manejar las excepciones de las sentencias SQL y manipular distintos descriptores de la base de datos.

**internet** “red de redes” global construida con base en protocolos estándar.

**interrelación** asociación entre tipos de entidades en el Modelo de Entidad Relación. En el Modelo Relacional, una interrelación es una conexión entre tablas mostradas por los valores de las columnas de una tabla que coinciden con los valores de las columnas de otra tabla. Las restricciones de integridad referencial y las llaves foráneas indican interrelaciones dentro del Modelo Relacional. Véase también relación uno-a-muchos, relación muchos-a-muchos e integridad referencial.

**interrelacionado** característica fundamental de las bases de datos. Interrelacionado significa que los datos almacenados en distintas unidades pueden relacionarse para proporcionar el cuadro completo. Para poder soportar las características interrelacionadas, la base de datos debe contener grupos de datos, conocidos como entidades, y relaciones que conecten las entidades.

**intersección** operador del álgebra relacional que combina las filas de dos tablas. El operador de intersección encuentra filas comunes en ambas tablas. Para poder utilizar el operador de intersección ambas tablas deben ser compatibles mediante un enlace.

**intranet** conjunto de computadoras y dispositivos de comunicación que utilizan el protocolo TCP/IP. Por razones de seguridad, generalmente a las computadoras de una intranet no se puede acceder desde Internet.

**ISO** Organización Internacional de Estándares, uno de los grupos responsables de los estándares SQL.

## J

**jerarquía de generalización** colección de tipos de entidad organizadas en una estructura jerárquica para mostrar las

semejanzas en los atributos. Cada subtipo o entidad hijo representa un subconjunto de su supertipo o entidad padre. Véase también supertipo y subtipo.

## L

**lectura no repetible** problema del control de concurrencias en el que una transacción lee el mismo valor más de una vez. Mientras se hace la lectura de un elemento de datos, otra transacción modifica el elemento.

**lenguaje de bases de datos no procedural** lenguaje tipo SQL que le permite especificar a qué parte de la base de datos acceder en lugar de tener que codificar un procedimiento complejo. Los lenguajes no procedurales no incluyen sentencias de ciclos.

**lenguaje de hipertexto marcado (HTML)** lenguaje en el que están escritos la mayoría de documentos web. HTML combina la estructura, el contenido y el formato del documento. Véase también XML y XSL.

**lenguaje de programación de base de datos** lenguaje procedural con una interfase hacia uno o más DBMSs. La interfase permite que un programa combine sentencias con procedimientos y accesos a bases de datos no procedurales. Véase también interfase de llamada-nivel e interfase sentencia-nivel.

**ley de transitividad** regla que establece que si un objeto A se relaciona con un objeto B y B se relaciona con C, entonces se puede concluir que A está relacionado con C. Las dependencias funcionales obedecen a la ley de transitividad. Véase también dependencia funcional y dependencia transitiva.

**ligas de consultas** proceso de asociar una consulta con un plan de acceso. Algunos DBMSs hacen las ligas de forma automática cuando una consulta o la base de datos cambia (estructuras de archivos, perfiles de tablas, tipos de datos, etc.).

**Línea de detalle** la línea más interna (más anidada) de un reporte jerárquico.

## LL

**llave candidata** superllave mínima. Una superllave es mínima si al eliminar cualquiera de sus columnas hace que ya no sea única.

**llave foránea** columna o combinación de columnas en la que los valores deben coincidir con aquéllos de la llave candidata. Una llave foránea debe tener el mismo tipo de datos de su llave candidata asociada. En la sentencia CREATE TABLE de SQL:2003, una llave foránea puede asociarse con una llave primaria en lugar de sólo con la llave candidata.

**llave primaria** llave candidata especialmente designada. La llave primaria de una tabla no puede contener valores nulos.

**llave primaria combinada** combinación de columnas (más de una) designadas como llave primaria. También se les conoce como llaves primarias compuestas.

## M

**mapeo de esquemas** método que describe cómo se deriva un esquema de un nivel más alto a partir de un esquema de nivel más bajo. Un mapeo proporciona el conocimiento para convertir la solicitud de una representación de un esquema más

alto hacia una representación de un esquema más bajo. Véase también arquitectura de los tres esquemas y esquema.

**materialización de vistas** método para procesar una consulta en una vista ejecutando directamente la consulta en la vista almacenada. La vista almacenada puede materializarse bajo demanda (cuando se envía la solicitud de consulta) o reconstruirla de forma periódica a partir de las tablas base. Para los almacenes de datos, se prefiere la estrategia de materialización para procesar las consultas de las vistas.

**mensaje** solicitud para invocar un método en un objeto. Cuando un objeto recibe un mensaje, busca una implementación en su propia clase. Si no se encuentra una implementación, el mensaje se envía a la clase padre del objeto. Véase también ligas.

**metadatos** datos que describen otros datos, incluidos los orígenes, uso, valor y significado de los mismos. Véase también diccionario de datos.

**middleware** componente de software en una arquitectura cliente-servidor que lleva a cabo la administración de procesos. El middleware permite que los servidores procesen mensajes de forma eficiente a partir de un gran número de clientes. Además, el middleware les permite a los clientes y servidores comunicarse entre plataformas heterogéneas. Los tipos más importantes de middleware de bases de datos incluyen monitores de procesamiento de transacciones, middleware orientado a mensajes y agentes de solicitud de objetos.

**middleware de acceso a datos** middleware que provee una interface uniforme hacia datos relacionales y no relacionales con el uso de SQL. Las consultas para acceder a datos a través de un manejador de bases de datos se envían a un controlador de acceso a datos en lugar de enviarlas de forma directa al DBMS. El controlador de acceso a datos convierte la sentencia SQL en SQL que entiende el DBMS y después dirige la consulta hacia el manejador. Los dos middleware de acceso a datos líderes son la Conectividad Abierta de Bases de Datos (ODBC), apoyada por Microsoft, y la Conectividad de Bases de Datos con Java (JDBC), apoyada por Oracle.

**middleware orientado a mensajes** middleware que conserva una cola de mensajes. Un proceso cliente puede colocar un mensaje en una cola y un servidor puede quitar el mensaje de la cola. El middleware orientado a mensajes soporta mensajes complejos entre clientes y servidores.

**middleware de bases de datos de objetos** arquitectura para bases de datos de objetos en la que el middleware administra datos complejos que posiblemente estén almacenados fuera de la base de datos, junto con los datos tradicionales almacenados en la base de datos.

**minería de datos** proceso mediante el cual se descubren patrones implícitos en los datos almacenados en un data warehouse y se utilizan dichos patrones para tener ventajas de negocio, tales como predecir tendencias futuras.

**modelo de cascada** marco de trabajo de referencia para el desarrollo de sistemas de información. El modelo de cascada está formado por iteración de análisis, diseño e implementación.

**modelo de datos corporativo (EDM)** modelo de datos conceptual de una organización. Un modelo de datos

corporativo puede usarse para planear los datos (lo que se desarrolla en una base de datos) y apoyo en la toma de decisiones (cómo integrar y transformar las bases de datos operacionales y las fuentes externas).

**modelo de datos** modelo gráfico que ilustra la estructura de una base de datos. Un modelo de datos contiene tipos de entidades y relaciones entre ellas. Véase también modelo de interacción de entorno y modelo del proceso.

**modelo de interacción con el entorno** modelo gráfico que muestra las relaciones entre eventos y procesos. Un evento, tal como el paso del tiempo o una acción desde el entorno, puede generar que un proceso inicie o se detenga. Véase también modelo de datos y modelo de procesos.

**modelo de maduración de un data warehouse** marco de trabajo que proporciona una guía de las decisiones de inversión en tecnología de data warehouses. El modelo está formado por seis etapas (prenatal, infantil, niñez, adolescente, adulto y sabio); el valor hacia el negocio aumenta mientras las organizaciones progresen hacia etapas más avanzadas.

**modelo de procesos** modelo gráfico que muestra las relaciones entre los procesos. Un proceso puede proporcionar datos de entrada utilizados por otros procesos o utilizar datos de salida de otros procesos. El diagrama de flujo ampliamente conocido es un ejemplo de un modelo de procesos. Véase también modelo de datos y modelo de interacción con el entorno.

**modelo relacional de datos** juego de valores para las conexiones entre tablas y operadores de tablas para representar un conjunto de datos.

**modificación de vistas** método para procesar una consulta en una vista que incluye la ejecución de una sola consulta. Una consulta que utiliza una vista se traduce en una consulta que usa las tablas base, reemplazando las referencias de la vista con su definición. La modificación de vistas es la estrategia preferida para el procesamiento de la mayoría de las consultas de bases de datos transaccionales.

**MOLAP** acrónimo de Procesamiento Analítico Multidimensional en Línea. MOLAP es un motor de almacenamiento que almacena y manipula los cubos de forma directa. Los motores MOLAP generalmente ofrecen el mejor desempeño de consultas posible, pero sufren la limitante del tamaño de cubos de datos que soportan.

**monitor de procesamiento de transacciones** uno de los primeros y aún importante middleware de bases de datos. Un monitor de procesamiento de transacciones recibe transacciones, las agenda, y las administra hasta que finalizan. Los monitores de procesamiento de transacciones también pueden apoyar la actualización de varias bases de datos en una sola transacción.

**MVD no trivial** MVD que no es FD. Por definición, cada FD es una MVD. Sin embargo, no todas las MVDs son FDs. Una MVD no trivial es aquella en la que una columna está asociada con más de un valor de dos columnas.

## N

**nivel de aislamiento** elemento que define el grado en el que una transacción se separa de las acciones de otras transacciones. Un diseñador de transacciones puede balancear la sobrecarga

del control de concurrencias con problemas de interferencia, prevenidos al especificar el nivel de aislamiento apropiado.

**nodo llave** campo(s) en un nodo de una estructura con valores únicos. La llave del nodo raíz es única entre todas las instancias. La llave de un nodo hijo es única dentro de su nodo padre.

**normalización** proceso mediante el cual se eliminan redundancias de las tablas para que sean más fáciles de modificar. Para normalizar una tabla, liste las dependencias funcionales y haga tablas que satisfagan una forma normal, generalmente la tercera forma normal (3NF) o la forma normal Boyce-Codd (BCNF).

## O

**objeto** instancia de una clase en la computación orientada a objetos. Un objeto tiene un identificador único que es invisible y no modificable.

**objeto binario grande (BLOB)** tipo de dato para campos que contienen datos binarios grandes, tales como imágenes. La información BLOB puede obtenerse, pero no puede mostrarse. El tipo de datos BLOB proporciona una forma sencilla de ampliar un manejador de bases de datos con funciones de objetos. Véase también la arquitectura de objetos grandes.

**objeto grande de caracteres (CLOB)** tipo de datos para las columnas que contienen demasiado texto, como documentos y páginas web. El tipo de datos CLOB proporciona una forma sencilla de ampliar un manejador de bases de datos con funciones de objetos. Véase también arquitectura grande de objetos.

**OLAP (procesamiento analítico en línea)** nombre general de la tecnología que soporta bases de datos multidimensionales. La tecnología OLAP cumple con el modelo de datos multidimensional y los alcances de implementación.

**oper (operational) mart** conjunto de datos del tipo justo-a-tiempo generalmente construido a partir de una base de datos operacional como anticipación o respuesta a eventos importantes, como los desastres y la introducción de nuevos productos. Un oper mart soporta la demanda pico para crear reportes y hacer los análisis de negocios que acompañan a un evento importante. Véase también data mart.

**operador BETWEEN-AND** operador abreviado para evaluar una columna numérica o de fechas dentro de un rango de valores. El operador BETWEEN-AND resulta verdadero si la columna es mayor o igual al primer valor y menor o igual al segundo valor.

**operador CUBE** operador que incrementa el resultado normal de un GROUP BY con todas las combinaciones de los subtotales. El estándar SQL:2003 proporciona el operador CUBE como una extensión de la cláusula GROUP BY para el soporte de datos multidimensionales. El operador CUBE es apropiado para resumir las columnas de varias dimensiones en lugar de columnas que representen distintos niveles de una sola dimensión.

**operador de semi-enlace** operador del álgebra relacional que es especialmente útil para el procesamiento distribuido de bases de datos. Un semi-enlace (semi-join) es la mitad de un enlace: las filas de una tabla que coinciden con al menos una fila de otra tabla.

**operador GROUPING SETS** operador en la cláusula GROUP BY que requiere de una especificación explícita de una combinación de columnas. El operador GROUPING SETS es apropiado cuando se requiere de un control sobre la agrupación. El estándar SQL:2003 proporciona el operador GROUPING SETS como una extensión de la cláusula GROUP BY para soportar datos multidimensionales.

**operador ROLL-UP** operador en la cláusula GROUP BY que aumenta el resultado normal del resultado GROUP BY con un subconjunto parcial de subtotales. El estándar SQL:2003 proporciona el operador CUBE como una ampliación de la cláusula GROUP BY para soportar datos multidimensionales. El operador ROLL-UP es apropiado para crear resúmenes de las columnas de una dimensión de la misma jerarquía.

**operadores tradicionales de conjuntos** operadores de unión, intersección y diferenciación del álgebra relacional.

## P

**paquete** unidad de modularidad PL/SQL. Los paquetes soportan una unidad más grande de modularidad que los procedimientos y funciones. Un paquete puede contener procedimientos, funciones, excepciones, variables, constantes, tipos y cursor. Al agrupar los objetos relacionados de forma conjunta, un paquete proporciona una reutilización más sencilla que los procedimientos individuales y funciones.

**perfil de aplicación** resumen estadístico de los formularios, reportes y consultas que acceden a una base de datos. Los perfiles de aplicación son una entrada importante del diseño físico de base de datos, ya que se utilizan para predecir la demanda de la base de datos.

**perfil de la tabla** resumen estadístico de las filas, columnas y relaciones que participan en una tabla. Los perfiles de tablas son un insumo importante de la fase del diseño físico de bases de datos, ya que se usan para predecir la fracción de una tabla a la que se accede en una consulta.

**periodo de tiempo de carga** diferencia entre el tiempo transaccional y el tiempo de carga. La identificación del periodo de tiempo de carga es una parte importante de la administración de la actualización de un data warehouse. Véase también periodo de tiempo válido.

**persistencia** característica fundamental de las bases de datos. Persistencia significa que los datos tienen un tiempo de vida mucho más largo que el de la ejecución de un programa de cómputo. Para ser persistentes, los datos deben residir en un almacén estable, como un disco magnético.

**pivote** operador de un cubo de datos en el que se reacomodan las dimensiones de un cubo de datos. Véase también cubo de datos.

**plan de acceso** árbol que codifica las decisiones de la estructura de archivos para acceder a tablas individuales, el orden de las tablas relacionadas y el algoritmo para relacionarlas. Los planes de acceso se generan mediante la optimización del componente para implementar las consultas enviadas por los usuarios.

**planeación de los sistemas de información** proceso de desarrollo de modelos de datos corporativos, procesos y roles organizacionales. La evaluación de la planeación de sistemas

de información evalúa la existencia de sistemas, identifica oportunidades para aplicar la tecnología de la información para obtener ventajas competitivas y la planeación de nuevos sistemas. También se conoce como planeación del sistema de negocio, ingeniería de sistemas de información y arquitectura de sistemas de información.

**polimorfismo** principio del cómputo orientado a objetos en el que un sistema de cómputo tiene la habilidad de escoger entre distintas implementaciones de un método. La implementación apropiada la selecciona el sistema (DBMS de objetos o lenguaje de programación orientado a objetos). El polimorfismo permite tener un vocabulario más pequeño de procedimientos y aumenta la capacidad de compartir código.

**problema de lectura fantasma** problema del control de concurrencias en el que una transacción ejecuta una consulta con condiciones de los registros, pero otra transacción inserta nuevas filas o modifica las filas existente mientras que la primera transacción continúa ejecutándose. La primera transacción reejecuta otra vez la consulta original, pero los resultados son distintos a los resultados de la primera ejecución. Las filas nuevas son fantasmas porque no existían para la primera ejecución de la consulta.

**procedimiento almacenado** conjunto de sentencias administradas por el DBMS. Los procedimientos almacenados amplían las capacidades de SQL. La mayoría de los DBMSs proporcionan un lenguaje propietario en el que pueden escribirse los procedimientos almacenados.

**procedimiento de ejecución de disparadores** método que especifica el orden de ejecución entre distintos tipos de disparadores, restricciones de integridad y sentencias de manipulación de la base de datos. Los procedimientos de ejecución de disparadores pueden ser complejos debido a que las acciones de un disparador pueden lanzar otros disparadores. Véase también disparadores que se traslanan.

**procedimiento de síntesis simple** conjunto de pasos para generar tablas de la forma BCNF usando una colección de dependencias funcionales. El procedimiento de la síntesis simple se limita a las estructuras de dependencias sencillas.

**procesamiento de transacciones** procesamiento eficiente y confiable de grandes volúmenes de trabajo repetitivo. Los DBMSs se aseguran de que los usuarios simultáneos no interfieran entre ellos y que las fallas no ocasionen que se pierda trabajo. Véase también transacción.

**procesamiento distribuido** método que permite la cooperación de computadoras geográficamente dispersas para proveer accesos a datos y otros servicios. Véase también arquitectura cliente-servidor.

**proceso de evaluación conceptual** secuencia de operaciones y de tablas intermedias utilizadas con el fin de obtener el resultado de una sentencia SELECT. El proceso de evaluación es conceptual porque la mayoría de los compiladores de SQL reducirán pasos para producir el resultado. El proceso de evaluación conceptual puede ayudarle inicialmente al entendimiento de la sentencia SELECT y a comprender problemas más complejos.

**proceso jerárquico analítico** técnica de la teoría de decisiones para evaluar problemas con objetivos múltiples. El proceso jerárquico analítico puede utilizarse para seleccionar y

evaluar DBMS, permitiendo la asignación sistemática de pesos a los requerimientos y puntuación a las características de los DBMS candidatos.

**producto cruz extendido** operador del álgebra relacional que combina dos tablas. El operador del producto cruz extendido (producto, para abreviar) construye una tabla a partir de dos tablas, formada de todas las posibles combinaciones de las filas y de una de cada una de las dos tablas de entrada.

**producto de la transacción** número de transacciones procesadas por un intervalo de tiempo. Es una medida del desempeño del procesamiento de transacciones. Por lo general, el producto de una transacción se reporta como transacciones por minuto.

**propiedades ACID** propiedades transaccionales incluidas en los manejadores de bases de datos. ACID es un acrónimo de atómicas, consistentes, aisladas y durables. Atómicas significa todo o nada. Consistentes significa que una base de datos no viola las restricciones de integridad antes o después de terminar una transacción. Aisladas significa que otras transacciones no pueden ver las actualizaciones hechas por una transacción sino hasta que haya terminado. Durable significa que los efectos de una transacción terminada son permanentes, incluso si ocurriera una falla.

**protocolo de asignación de dos fases (2PC)** regla para asegurar que las transacciones distribuidas sean atómicas. 2PC utiliza una fase de votos y otra de decisión para coordinar los compromisos de las transacciones locales.

**protocolo de bloqueo de dos fases (2PL)** regla para asegurar que las transacciones concurrentes no interfieran entre ellas. 2PL requiere que los bloqueos se usen antes de leer o escribir un elemento de la base de datos. Una transacción espera para saber si existe un bloqueo sobre el elemento de la base de datos y no se liberan los bloqueos hasta que ya no se requieran. Para simplificar la implementación, la mayoría de los DBMS conservan al menos bloqueos exclusivos hasta que termina la transacción.

**protocolo de copia primaria** protocolo para el control de concurrencias de las transacciones distribuidas. A cada fragmento que se copia se le designa como copia principal o copia secundaria. Durante el procesamiento distribuido de transacciones, sólo se garantiza que al final de la transacción la copia principal sea la actual. Las actualizaciones pueden propagarse hacia las copias secundarias después de haber terminado la transacción.

**protocolo de escritura de bitácora** protocolo con el cual, en el proceso de actualización inmediata, los registros de la bitácora deben escribirse en un almacenamiento estable antes de los registros correspondientes de la base de datos.

**prototipo** implementación rápida de una aplicación en un sistema de información. Los prototipos pueden demostrar formularios, reportes y menús para habilitar la retroalimentación por parte de los usuarios.

**proyecto** operador del álgebra relacional. Una operación de un proyecto extrae un subconjunto de columnas especificadas en la tabla de entrada. Las filas duplicadas en el resultado se eliminan.

**pruebas de referencia (benchmark)** carga de trabajo para evaluar el desempeño de un sistema o producto. Una buena prueba debe ser relevante, portátil, escalable y entendible.

**punto de guardado (save point)** punto intermedio de una transacción en el que puede ocurrir un retorno (rollback). Los puntos de guardado se incluyen mediante ampliaciones propietarias al estándar SQL y el estándar SQL:2003.

**punto de revisión (checkpoint)** acto en el que se escribe un registro de punto de revisión en la bitácora y con el que se obliga a escribir el registro de la bitácora y de los búferes de la base de datos en el disco. Todas las actividades transaccionales deben cesar cuando se llegue a un control. El intervalo de un punto de revisión debe escogerse de tal forma que balancee el tiempo de reinicio con la sobrecarga del control. Un punto de revisión tradicional es el conocido como punto de revisión de consistencia del caché. Véase también punto de revisión difuso y punto de revisión incremental.

**punto de revisión difuso** alternativa a los controles tradicionales de caché consistentes que involucra menor carga pero que puede requerir más trabajo de reinicio. En un punto de revisión difuso, el administrador de recuperaciones solamente escribe las páginas del búfer desde el punto de revisión previo. La mayoría de estas páginas debieron haberse escrito en el disco antes del punto de revisión. En el momento de reiniciar, el administrador de recuperaciones usa los dos registros más recientes de revisión difusa de la bitácora. Véase también punto de revisión.

**punto muerto (deadlock)** problema de espera mutua que ocurre cuando se usan candados. Si un punto muerto no se resuelve, las transacciones involucradas estarán en espera indefinidamente.

## R

**RAID (arreglos redundantes de discos independientes)** colección de discos (arreglo de discos) que operan como uno solo. El almacenamiento RAID soporta operaciones de lectura y escritura en paralelo y con alta confiabilidad.

**RAID-1** arquitectura del almacenamiento RAID en la que los arreglos redundantes de discos proporcionan alta confiabilidad y desempeño pero con una gran sobrecarga de almacenamiento. RAID-1 usa discos en espejo para conseguir su alto desempeño y confiabilidad.

**RAID-5** arquitectura del almacenamiento RAID en la que las páginas de corrección de errores se ubican de forma aleatoria para proporcionar una alta confiabilidad sin una sobrecarga excesiva de almacenamiento. RAID-5 usa la división para conseguir su alto desempeño y confiabilidad sin sobrecarga excesiva de almacenamiento.

**rebanada** operador de un cubo de datos en el que una dimensión es reemplazada por un valor único o por un resumen de sus valores miembro. Véase también dado.

**red de área de almacenamiento (SAN)** red especializada de alta velocidad que conecta los dispositivos de almacenamiento y los servidores. El objetivo de la tecnología SAN es integrar distintos tipos de subsistemas de almacenamiento en un solo sistema y eliminar los cuellos de botella de un servidor único que controle los dispositivos de almacenamiento. Una SAN es el complemento para el almacenamiento de discos RAID.

**reescritura de consultas** proceso de sustitución en el que una vista materializada reemplaza las referencias hacia las

tablas de hechos y dimensiones de una consulta. Además de realizar la sustitución, el optimizador de consultas evalúa si la sustitución mejorará el desempeño de la consulta original. Véase también vista materializada.

**registro en uso** datos comunes que los distintos usuarios intentan modificar al mismo tiempo. Sin los controles de accesos adecuados, un usuario puede interferir con otro en el registro en uso. Un registro en uso, independiente del sistema, no depende de los detalles de un control administrativo de accesos en particular. Por lo general, un registro en uso independiente del sistema incluye campos o filas de una base de datos. Un registro de uso independiente del sistema depende de los detalles del control administrativo de accesos, en especial por la granularidad bloqueada.

**registro físico** colección de bytes que se transfieren entre el almacenamiento volátil de la memoria principal y el almacenamiento estable en disco. El número de accesos a los registros físicos es una medida importante del desempeño de una base de datos.

**reglas de autorización** reglas que definen a los usuarios autorizados las operaciones permitidas y las partes de una base de datos a las que puede accederse. El sistema de seguridad de la base de datos almacena las reglas de autorización y las aplica para cada acceso a la base de datos. También se conocen como restricciones de seguridad.

**reglas sobre las filas referenciadas** reglas que describen las acciones hechas sobre las filas referenciadas cuando la fila de la llave primaria de una tabla (fila referenciada) se elimina o actualiza su llave primaria.

**relación** sinónimo de tabla. Un término que generalmente se usa en la investigación académica sobre bases de datos.

**relación autorreferenciada** relación que involucra a la misma tabla o tipo de entidad. Las relaciones autorreferenciadas representan asociaciones entre miembros del mismo conjunto. También se les conoce como relaciones unitarias, reflexivas o recursivas.

**relación identificadora** relación que proporciona el componente de la llave primaria a una entidad débil. Véase también entidad débil y dependencia identificadora.

**relación muchos-a-muchos (M-N)** relación en el modelo de entidad relación, en la que los objetos de cada tipo de entidad pueden relacionarse con muchos objetos de otro tipo de entidad. Las relaciones M-N tienen cardinalidades máximas de más de uno en cada dirección. En el Modelo Relacional, dos relaciones 1-M y una tabla de enlace o asociativa representan relaciones M-N. Véase también relación uno-a-muchos y relación.

**relación M-way (multiforma)** relación que involucra a más de dos tipos de entidad. En algunas notaciones de ERD (como la de la pata de cuervo), una relación M-way se representa como un tipo de entidad asociativo M-way. Un tipo de entidad asociativo M-way tiene más de dos relaciones identificables.

**relación obligatoria** relación con una cardinalidad mínima de uno o más. Una relación obligatoria genera una dependencia sobre el tipo de entidad asociado con la cardinalidad mínima de uno. Véase también relación opcional y dependencia de existencia.

**relación opcional** relación con una cardinalidad mínima de cero. Una relación opcional significa que las entidades pueden almacenarse sin participar en la relación. Véase también relación obligatoria.

**relación ternaria** relación que incluye tres tipos de entidad. En algunas notaciones de los ERD, como la notación de la pata de cuervo, una relación ternaria se representa como un tipo de entidad asociativo con tres relaciones 1-M.

**relación uno-a-muchos (1-M)** relación en el modelo entidad-relación en la que la cardinalidad máxima es de 1 en una dirección y de M en la otra dirección. En el modelo de datos relacional, una restricción de integridad referencial generalmente indica una relación 1-M. Véase también relación y relación muchos a muchos.

**reporte** presentación estilizada de los datos apropiados para una audiencia determinada. Los reportes mejoran la apariencia de los datos que se despliegan o imprimen. Véase también reporte jerárquico.

**reporte jerárquico** despliegue con formato de una consulta usando formularios para mostrar agrupación y orden. También se conocen como reportes de corte.

**restricción de completez** restricción de las jerarquías de generalización. Una restricción de este tipo significa que cada entidad en un supertipo tiene una entidad relacionada en uno de los subtipos. En otras palabras, la unión del conjunto de entidades en los subtipos es igual al conjunto de entidades del supertipo.

**restricción de desunión** restricción de las jerarquías de generalización. Una restricción de desunión significa que los subtipos no comparten ninguna entidad en común. En otras palabras, la intersección de los conjuntos de entidades en los subtipos está vacía.

**restricción para refrescar** restricción de un data warehouse o sistema origen que limita los detalles de un proceso para refrescar datos. Las restricciones para refrescar pueden clasificarse como acceso fuente, integración, disponibilidad de un data warehouse o consistencia de integridad.

**restringir** operador del álgebra relacional. Una operación de restricción extrae un subconjunto de filas que satisfacen una condición dada.

**resumen** operador del álgebra relacional que comprime las filas de una tabla. Una operación de resumen produce una tabla con filas que resumen las filas de la tabla inicial. Las funciones agregadas se usan para resumir las filas de la tabla inicial.

**resumen incorrecto** problema del control de concurrencias en el cual la transacción lee varios valores, pero alguna otra transacción actualiza alguno de los valores mientras la primera transacción sigue ejecutándose. Se le conoce también como análisis inconsistente.

**retraso válido de tiempo** diferencia entre la ocurrencia de un evento en el mundo real (tiempo válido) y el almacenamiento del evento en una base de datos operacional (tiempo de transacción). Véase también tiempo válido, retraso de tiempo de la carga y tiempo de transacción.

**revisión de restricciones diferidas** restricciones que obligan a mantener la integridad al final de una transacción en lugar de hacerlo de forma inmediata después de cada sentencia

manipulada. Las restricciones complejas pueden beneficiarse de la revisión diferida.

**revisión de tipos fuertes** habilidad para asegurarse de que las expresiones no contienen errores de compatibilidad. La revisión de tipos fuertes es una categoría importante de revisión de errores para la codificación de objetos.

**ROLAP** acrónimo para Procesamiento Relacional Analítico en Línea. ROLAP incluye ampliaciones a un DBMS relacional para soportar datos multidimensionales. Los motores ROLAP soportan una gran variedad de técnicas de almacenamiento y organización para la extracción de datos en forma de resumen.

## S

**segunda forma normal (2NF)** característica de una tabla; una tabla se encuentra en 2NF si cada columna no-llave depende de una llave completa, y no de parte de una llave.

**seguridad de la base de datos** mecanismo que protege a las bases de datos de accesos no autorizados y destrucciones maliciosas.

**selección del índice** acción mediante la cual se elige al menos un índice agrupado y cero o más índices desagrupados para cada tabla. En un índice agrupado, el orden de los datos de los registros es parecido al orden del índice. En un índice desagrupado, el orden de los datos de los registros no se relaciona con el orden del índice. La selección del índice es un subproblema importante del diseño físico de bases de datos. La selección del índice comúnmente opta por los índices Btree. Los otros índices (hash y mapa de bits) también pueden ser tomados en cuenta.

**servidor** programa que procesa solicitudes a nombre de un cliente. Un servidor de base de datos puede interpretar sentencias SQL, ubicar datos, actualizar tablas, revisar restricciones de integridad y devolver datos a los clientes.

**sinónimo** grupo de palabras que se deletrean diferente pero que tienen el mismo significado, en la vista de integración. Los sinónimos generalmente se observan cuando diferentes partes de una organización pueden usar un vocabulario distinto para describir las mismas cosas. Véase también homónimo.

**sistema** conjunto de componentes relacionados que trabajan de forma conjunta para cumplir con determinados objetivos.

**sistema de administración de base de datos (DBMS)** colección de componentes que apoyan la adquisición, diseminación, mantenimiento, obtención y formateo de datos. Un DBMS corporativo soporta las bases de datos críticas de una organización. Un DBMS de escritorio soporta las bases de datos para pequeños grupos de trabajo y pequeños negocios. Un DBMS embebido forma parte de un sistema más grande, como un dispositivo o una aplicación. Los DBMS embebidos proporcionan funciones limitadas, pero tienen requerimientos de poca memoria, procesamiento y almacenamiento.

**sistema de administración de bases de datos paralelas**

**(DBMS)** DBMS capaz de utilizar recursos de cómputo altamente acoplados (procesadores, discos y memoria). El acoplamiento se consigue mediante redes con un tiempo de intercambio de datos comparable al tiempo de intercambio de datos con un disco. La tecnología de bases de datos paralelas promete mejoras en el desempeño y la alta disponibilidad,

pero con problemas de interoperabilidad si no se administra de forma apropiada. Véase también aceleración y escalabilidad.

**sistema de información** sistema que acepta datos desde su entorno, los procesa y genera su impresión para la toma de decisiones. Un sistema de información está formado por personas, procedimientos, impresión de datos, salida de datos, bases de datos, software y hardware.

**sort merge** algoritmo de enlace que requiere que las dos tablas se encuentren ordenadas por la columna de enlace. El algoritmo sort merge sólo puede usarse para los equienlaces. El algoritmo sort merge se desempeña bien si el costo de la clasificación es pequeño o si existe un índice de enlace agrupado. Véase también enlace hash y ciclos anidados.

**SQL** acrónimo para Lenguaje de Consultas

Estructuradas. SQL es un estándar de la industria de los lenguajes de bases de datos que incluye sentencias para la definición de bases de datos (como la sentencia CREATE TABLE), manipulación de bases de datos (como la sentencia SELECT) y control de las bases de datos (como la sentencia GRANT). SQL comenzó como un lenguaje propietario desarrollado por IBM. Ahora SQL está ampliamente soportado por el estándar internacional de bases de datos.

**SQL aislado** uso de un editor especializado que envía las sentencias SQL directamente al DBMS y despliega los resultados devueltos por el DBMS. Véase también SQL embebido.

**SQL embebido** uso de SQL dentro de un lenguaje de programación, como COBOL o Visual Basic. Sentencias SQL adicionales que se pueden usar solamente dentro de un lenguaje de programación que ocasionan que otras sentencias SQL se ejecuten y usen los resultados de una base de datos dentro de un programa. Véase también SQL único.

**SQL:2003** el estándar más reciente del Lenguaje de Consultas Estructuradas. SQL:2003 soporta muchas ampliaciones que están más allá de SQL-92 y de las características de actualización que se especificaron en el estándar previo (SQL:1999). El estándar SQL:2003 incluye nueve partes y siete paquetes. El núcleo de SQL:2003 está formado por las partes 1, 2 y 11. Cada parte que no forma parte del núcleo contiene características obligatorias y opcionales. Un paquete es una colección de características opcionales para algún área de una aplicación o entorno de implementación.

**subconsulta** véase consulta anidada.

**subformulario** variable o remanente de un formulario jerárquico. El subformulario puede mostrar varios registros al mismo tiempo.

**subtipo** tipo de entidad hijo en una jerarquía de generalización. Un subtipo representa un tipo de entidad más especializada que un supertipo.

**superllave** columna o combinación de columnas que contienen valores únicos para cada renglón. La combinación de cada columna de una tabla siempre es una superllave, ya que las filas de una tabla deben ser únicas.

**supertipo** tipo de entidad padre en una jerarquía de generalización. Un supertipo representa un tipo de entidad más general que sus subtipos.

**supuesto de valor uniforme** valor por el cual se asume que el valor de cada columna es parecido (tiene el mismo número de filas). El supuesto del valor uniforme permite la representación compacta de una distribución, pero puede conducir a grandes errores de estimación que conduzcan a malas elecciones en la optimización de consultas y selección de índices.

## T

**tabla** arreglo bidimensional de datos. Una tabla está formada por la parte del encabezado y la parte del cuerpo.

**tabla de hechos** tabla en un esquema de estrella o esquema de copos de nieve que almacena valores numéricos relevantes para la persona que toma decisiones. Véase también esquema de estrella y esquema de copos de nieve.

**tabla que conserva las llaves** término de Oracle para una tabla actualizable en una vista de enlaces. Una vista de enlaces conserva una tabla cuando cada llave candidata de la tabla puede ser una llave candidata del enlace resultante. Esto significa que las filas de una vista de un enlace actualizable pueden relacionarse de manera 1-1 con cada una de las llaves conservadas. En un enlace que involucra una relación 1-M, la tabla hijo puede conservarse porque cada renglón hijo se asocia con al menos un renglón padre.

**tablas de dimensiones** tabla en un esquema de estrella o de copo de nieve que almacena las dimensiones o temas utilizados para agregar hechos.

**tecnología de base de datos de cuarta generación**

tecnología que extiende las fronteras de la tecnología de base de datos hacia datos poco convencionales y el Internet. Los sistemas de cuarta generación pueden almacenar y manipular tipos de datos poco convencionales, como imágenes, videos, mapas, sonidos y animaciones, así como proporcionar acceso web a bases de datos. La tecnología de bases de datos de cuarta generación se comercializó ampliamente durante los años noventa.

**tecnología de base de datos de primera generación**

estructuras de archivos propietarias e interfaces de programas que soportan las búsquedas secuenciales y aleatorias. Sin embargo, al usuario se le solicita que escriba un programa de cómputo para acceder a los datos. La tecnología de bases de datos de primera generación fue ampliamente desarrollada durante los años sesenta.

**tecnología de bases de datos de segunda generación** el primer DBMS que realmente administró múltiples tipos de entidades y relaciones. Sin embargo, para obtener acceso a los datos, todavía tenía que escribirse un programa de cómputo. La tecnología de bases de datos de segunda generación se desarrolló ampliamente durante los años setenta.

**tecnología de tercera generación de bases de datos**

DBMS relacionales que incorporan accesos sin procedimientos, tecnología de optimización y capacidades de procesamiento de transacciones. La tecnología de bases de datos de tercera generación se comercializó ampliamente durante los años ochenta.

**tercera forma normal (3NF)** característica de una tabla. Una tabla se encuentra en 3NF si está en 2NF y cada columna no-llave depende de sólo una llave.

**tiempo de carga** tiempo en el que se actualiza un data warehouse.

**tiempo de transacción** tiempo que tarda en actualizarse una fuente de datos operacional.

**tiempo válido** tiempo en que ocurre un evento.

**tipo de datos** colección de datos que define un conjunto de valores y operaciones permisibles. Cada columna de una tabla se asocia con un tipo de dato.

**tipo de entidad** colección de entidades (personas, lugares, eventos o cosas) que son del interés de una aplicación, representada por un rectángulo en un diagrama de entidad relación.

**tipo de entidad asociativa** entidad débil que depende de dos o más tipos de entidades para su llave primaria. A un tipo de entidad asociativa con más de dos relaciones identificadas se le conoce como tipo de entidad asociativa M-way. Véase también relación M-way, relación identificada y relación débil.

**transacción** unidad de trabajo que debe procesarse de forma confiable. Los DBMSs proporcionan servicios para la recuperación y control de concurrencias para procesar las transacciones de forma eficiente y confiable.

**transparencia de la concurrencia** servicio proporcionado por el DBMS para que los usuarios perciban la base de datos como un sistema de usuario único, aunque haya múltiples usuarios simultáneos. El administrador del control de concurrencia es el componente del DBMS responsable de la transparencia de la concurrencia.

**transparencia de la fragmentación** nivel de independencia de datos en DBMSs distribuidos en el que las consultas pueden formularse sin conocer los fragmentos. Véase también transparencia de la ubicación y transparencia del mapeo local.

**transparencia de recuperaciones** servicio proporcionado por un DBMS para restaurar de forma automática una base de datos a un estado consistente después de una falla. El administrador de recuperaciones es el componente de un DBMS responsable de la transparencia de recuperaciones.

**transparencia local de mapas** nivel de independencia de datos en un DBMS distribuido en el que las consultas pueden formularse sin tener conocimiento de los formularios locales. Sin embargo, es necesario el conocimiento y la colocación de los fragmentos. Véase también fragmento, transparencia de fragmentos y ubicación de la transparencia.

## U

**ubicación de la transparencia** nivel de independencia de datos en un DBMS distribuido en el que las consultas pueden formularse sin tener conocimiento de las ubicaciones. Sin embargo, es necesario el conocimiento de los fragmentos. Véase también fragmento, transparencia de fragmentos y ubicación de la transparencia.

**unión** operador del álgebra relacional que combina las filas de dos tablas. El resultado de una operación de unión tiene todas las filas de las dos tablas. Las dos tablas deben ser compatibles con respecto al operador unión.

**usuario indirecto** usuario que accede a la base de datos a través de reportes o extractos de datos en lugar de hacerlo por

su propia iniciativa. Véase también usuario parametrizable y usuarios poderosos.

**usuario parametrizable** persona que usa una base de datos solicitando formularios y reportes existentes mediante el uso de parámetros, valores de entrada que se modifican entre un uso y otro. Véase también usuario indirecto y usuario total.

**usuario total** persona que usa una base de datos al enviar solicitudes de datos no planeadas o *ad hoc*. Los usuarios totales deben comprender lo que es un acceso sin procedimientos. Véase también usuario indirecto o usuario parametrizable.

## V

**valor nulo** valor especial que representa la ausencia de un valor real. Un valor nulo puede significar que el valor real es desconocido o no se utiliza en una fila dada.

**vista** tabla virtual o derivada. Una vista se deriva de tablas base o físicas a través de una consulta. Véase también vista materializada.

**vista actualizable** vista que puede usarse en las sentencias SELECT, UPDATE, INSERT y DELETE. Cuando se modifican las filas de una vista actualizable, el DBMS traduce las modificaciones de la vista en modificaciones de las filas de las tablas base.

**vista de sólo lectura** vista que puede utilizarse en las sentencias SELECT pero no en las sentencias UPDATE, INSERT y DELETE. Todas las vistas son al menos de sólo lectura.

**vista externa** descripción de los datos derivados de un grupo de usuarios dado. También se le conoce como esquema externo y vista. Véase también esquema y arquitectura de los tres esquemas.

**vista materializada** vista almacenada que debe sincronizarse de forma periódica con su fuente de datos. Los DBMSs relacionales proporcionan vistas materializadas con datos de tipo resumen para obtener respuesta rápida a las consultas. Véase también reescritura de consultas y vista.

## W

**WITH CHECK OPTION** cláusula de la sentencia CREATE VIEW que puede usarse para evitar actualizaciones con efectos colaterales. Si se especifica WITH CHECK OPTION, se rechazan las sentencias INSERT o UPDATE que violen la cláusula WHERE de una vista.

**world wide web (WWW)** conjunto de páginas que pueden verse en Internet. En la WWW, un navegador despliega las páginas enviadas por un servidor web. La WWW es la aplicación más popular de Internet.

## X

**XML (lenguaje de marcaje extendido)** lenguaje simple que soporta la especificación de otros lenguajes. XML ha evolucionado en un conjunto de lenguajes que separan el contenido, la estructura y el formato de documentos en la red mundial. El lenguaje de esquemas XML, miembro importante de la familia de lenguajes de XML, soporta la estandarización de la estructura de los documentos XML.

# Bibliografía

---

- Batini, C., Ceri, S., y Navathe, S. *Conceptual Database Design*, Redwood City, CA: Benjamin/Cummings, 1992.
- Batra, D. "A Method for Easing Normalization of User Views," *Journal of Management Information Systems* 14, 1 (verano 1997), 215-233.
- Bernstein, P. "Middleware: A Model for Distributed Services," *Communications of the ACM* 39, 2 (febrero 1996), 86-97.
- Bernstein, P. "Repositories and Object-Oriented Databases," en *Proceedings of BTW 97*, Ulm, Alemania, Springer-Verlag, (1997), pp. 34-46 (reimpreso en *ACM SIGMOD Record* 27 (1), marzo 1998).
- Bernstein, P. y Dayal, U. "An Overview of Repository Technology," en *Proceedings of the 20th Conference on Very Large Data Bases*, San Francisco: Morgan Kaufman, Agosto 1994, pp. 705-713.
- Bernstein, P. y Newcomer, E. *Principles of Transaction Processing*, San Francisco: Morgan Kaufmann, 1997.
- Booch, G., Jacobson, I., y Rumbaugh, J. *The Unified Modeling Language User Guide*, Reading, MA: Addison-Wesley, 1998.
- Bouzeghoub, M., Fabret, F., y Matulovic-Broque, M. "Modeling Data Warehouse Refreshment Process as a Workflow Application," en *Proceedings on the International Workshop on Design and Management of Data Warehouses*, Heidelberg, Alemania (junio 1999).
- Bowman, J., Emerson, S., y Darnovsky, M. *The Practical SQL Handbook*, 4a. ed., Reading, MA: Addison-Wesley, 2001.
- Carlis, J. and Maguire, J. *Mastering Data Modeling*, Reading, MA: Addison-Wesley, 2001.
- Castano, S., Figini, M., Giancarlo, M., y Pierangela, M. *Database Security*, Reading, MA: Addison-Wesley, 1995.
- Celko, J. *Joe Celko's SQL Puzzles & Answers*, San Francisco: Morgan Kaufmann, 1997.
- Ceri, S., y Pelagatti, G. *Distributed Databases: Principles and Systems*, Nueva York: McGraw-Hill, 1984.
- Chaudhuri, S. "An Overview of Query Optimization in Relational Systems," en *Proceedings of the ACM Symposium on Principles of Database Systems*, Seattle, WA, 1998, pp. 34-43.
- Chaudhuri, S., y Narasayya, V. "An Efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server," en *Proceedings of the 23rd VLDB Conference*, Atenas, Grecia, 1997, pp. 146-155.
- Chaudhuri, S., y Narasayya, V. "Automating Statistics Management for Query Optimizers," *IEEE Transactions on Knowledge and Data Engineering* 13, 1 (enero/febrero 2001), 7-28.
- Choobineh, J., Mannino, M., Konsynski, B., y Nunamaker, J. "An Expert Database Design System Based on Analysis of Forms," *IEEE Trans. Software Engineering* 14, 2 (febrero 1988), 242-253.
- Choobineh, J., Mannino, M., y Tseng, V. "A Form-Based Approach for Database Analysis and Design," *Communications of the ACM* 35, 2 (febrero 1992), 108-120.
- Codd, T. "A Relational Model for Large Shared Data-Banks," *Communications of the ACM* 13, 6 (junio-1970).
- Date, C. "What Is a Distributed Database System?" en *Relational Database Writings 1985-1989*, C. J. Date (ed.), Reading, MA: Addison-Wesley, 1990.
- Date, C. *Introduction to Database Systems*, 8a. ed., Reading, MA: Addison-Wesley, 2003.
- Date, C., and Darwen, H. *A Guide to the SQL Standard*, Reading, MA: Addison-Wesley, 1997.
- Davis, J. "Universal Servers: Part 1 y Part 2," *DBMS Online*, junio y julio 1997, <http://www.dbmsmag.com/9706d13.html>.
- Eckerson, W. "Gauge Your Data Warehouse Maturity," *DM Review*, noviembre 2004, [www.dmrreview.com](http://www.dmrreview.com).
- Elmasri, R., y Navathe, S. *Fundamentals of Database Systems*, 4a. ed., Redwood City, CA: Addison-Wesley, 2004.
- Fagin, R. "A Normal Form for Relational Databases That-Is Based on Domains and Keys," *ACM Transactions on Database Systems* 6, 3 (septiembre 1981), 387-415.
- Finkelstein, S., Schkolnick, M., y Tiberio, T. "Physical Database Design for Relational Databases," *ACM Transactions on Database Systems* 13, 1 (marzo 1988), 91-128.
- Fisher, J., y Berndt, D. "Creating False Memories: Temporal Reconstruction Errors in Data Warehouses," en *Proceedings Workshop on Technologies and Systems (WITS 2001)*, Nueva Orleans, diciembre 2001.
- Fowler, M., y Scott, K. *UML Distilled*, Reading, MA: Addison-Wesley, 1997.
- Graefe, G. "Options for Physical Database Design," *ACM SIGMOD Record* 22, 3 (septiembre 1993), 76-83.
- Gray, J., y Reuter, A. *Transaction Processing: Concepts-and Techniques*, San Francisco: Morgan Kaufmann, 1993.
- Groff, J., y Weinberg, P. *SQL: The Complete Reference*, 2nd ed., Nueva York: Osborne/McGraw-Hill, 2002.
- Gulutzan, P., y Pelzer, T. *SQL-99 Complete, Really*, Lawrence, KS: R & D Books, 1999.
- Hawryszkiewycz, I. *Database Analysis and Design*, Nueva-York: SRA, 1984.
- Imhoff, C. "Intelligent Solutions: Oper Marts: An Evolution in the Operational Data Store," *DM Review* 11, 9 (septiembre 2001), 16-18.

- Inmon, W. *Building the Data Warehouse*, 3a. ed., Nueva-York: John Wiley & Sons, 2002.
- Inmon, W. *Information Systems Architecture*, Nueva York: John Wiley & Sons, 1986.
- Jarke, M., and Koch, J. "Query Optimization in Database Systems," *ACM Computing Surveys* 16, 2 (junio-1984), 111-152.
- Kent, W. "A Simple Guide to the Five Normal Forms in Relational Database Theory," *Communications of the ACM* 26, 2 (febrero 1983), 120-125.
- Kimball, R. "Slowly Changing Dimensions," *DBMS* 9, 4-(abril 1996) 18-22.
- Kimball, R. *The Data Warehouse Toolkit*, 2a. ed., Nueva-York: John Wiley & Sons, 2002.
- Kimball, R. "The Soul of the Data Warehouse, Part 3: Handling Time," *Intelligent Enterprise Magazine*, abril-2003, <http://www.intelligententerprise.com>.
- Mannino, M., Chu, P., y Sager, T. "Statistical Profile Estimation in Database Systems," *ACM Computing Surveys* 20, 3 (septiembre 1988), 191-221.
- Martin, J. *Strategic Data-Planning Methodologies*, Englewood Cliffs, NJ: Prentice Hall, 1982.
- Melton, J., y Simon, A. *SQL:1999 Understanding Relational Language Components*, San Mateo, CA: Morgan-Kaufman, 2001.
- Melton, J., y Simon, A. *Understanding the New SQL: A Complete Guide*, San Mateo, CA: Morgan-Kaufman, 1992.
- Muller, R. *Database Design for Smarties: Using UML for Data Modeling*, San Francisco, CA: Morgan Kaufmann, 1999.
- Mullin, C. *Database Administration: The Complete Guide to Practices and Procedures*, Reading, MA: Addison Wesley, 2002.
- Nelson, M., y DeMichiel, L. "Recent Trade-Offs in SQL3," *ACM SIGMOD Record* 23, 4 (diciembre 1994), 84-89.
- Nijssen, G., y Halpin, T. *Conceptual Schema and Relational Database Design*, Sydney: Prentice Hall de Australia, 1989.
- Olson, J. *Data Quality: The Accuracy Dimension*, Nueva York: Morgan Kaufmann, 2002.
- Orfali, R., Harkey, D., y Edwards, J. *The Essential Client/Server Survival Guide*, 2a. ed., Nueva York: John Wiley, 1996.
- Ozsu, T., y Valduriez, P. *Principles of Distributed Database Systems*, Englewood Cliffs, NJ: Prentice Hall, 1991.
- Park, C., Kim, M., y Lee, Y. "Finding an Efficient Rewriting of OLAP Queries Using Materialized Views in Data Warehouses" *Decision Support Systems* 32, 12 (2002), 379-399.
- Peinl, P., Reuter, A., y Sammer, H. "High Contention in a Stock Trading Database: A Case Study," en *Proceedings of the ACM SIGMOD Conference*, Chicago, IL (mayo 1988), pp. 260-268.
- Redman, T. *Data Quality: The Field Guide*, Nueva York: Digital Press, 2001.
- Saaty, T. *The Analytic Hierarchy Process*, Nueva York: McGraw-Hill, 1988.
- Shasha, D., y Bonnet, P. *Database Tuning: Principles, Experiments, and Troubleshooting Techniques*, San Francisco: Morgan Kaufmann, 2003.
- Sheth, A., Georgakopoulos, D., y Hornrick, M. "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure," *Distributed and Parallel Databases* 3, (1995), 119-153, Kluwer Academic Publishers.
- Sigal, M. "A Common Sense Development Strategy," *Communications of the ACM* 41, 9 (septiembre 1998), 42-48.
- Su, S., Dujmovic, J., Batory, D., Navathe, S., y Elnicki, R. "A Cost-Benefit Decision Model: Analysis, Comparison, and Selection of Data Management Systems," *ACM Transactions on Database Systems* 12, 3 (septiembre 1987), 472-520.
- Sutter, J. "Project-Based Warehouses," *Communications of the ACM* 41, 9 (septiembre 1998), 49-51.
- Teorey, T. *Database Modeling and Design*, 3a. ed., San-Francisco: Morgan Kaufman, 1999.
- Watson, R. *Data Management: An Organizational Perspective*, Nueva York: John Wiley and Sons, 1996.
- Whitten, J., y Bentley, L. *Systems Analysis and Design Methods*, Nueva York: Irwin/McGraw-Hill, 2001.
- Zahedi, F. "The Analytic Hierarchy Process: A Survey of the Method and Its Applications," *Interfaces* 16, 4, (1986), 96-108.

# Índice

---

## Números y símbolos

% (símbolo de porcentaje), carácter comodín, 87  
\*(asterisco) carácter comodín, 85, 86, 87-88  
\_ (guion bajo), que concuerda con cualquier carácter de un dígito, 88  
(+) notación como parte de una condición de intersección, 335-337  
:= (asignación) símbolo de, 381, 382  
? (signo de interrogación), que concuerda con cualquier carácter de un dígito, 88  
; (punto y coma)  
  en PL/SQL, 380  
  en SQL \*Plus, 386  
(diagonal) en SQL \*Plus, 386  
1-1, relaciones de tipo, 139, 191-193  
I-M, consultas que se pueden actualizar, 350, 688  
  con más de dos tablas, 352-353  
  en Access, 350-351  
   inserción de registros en, 351-352

## A

acceso a base de datos de Web, 378  
acceso de aplicaciones de bases de datos para Student Loan Limited, 464  
acceso de datos  
  código, 610  
  controlador, 611, 681  
  herramientas para la minería de datos, 559  
  software personalizado, 611, 681  
acceso de discos, velocidad de, 251  
acceso de memoria, velocidad de, 251  
acceso de procedimiento, 8  
acceso simultáneo para las bases de datos, 520-522  
acceso sin procedimiento, 7, 8-9  
  en tercera generación de DBMS, 12  
  soportados por objetos del DBMS, 649  
accesos a registros físicos  
  en las operaciones Btree, 261  
  dificultad de predicción, 253  
  que limitan el desempeño, 251  
Access. *Vea* Microsoft Access  
acción de nulificar, 55, 679  
acción de restricción, 679  
acción en cascada, 679  
acción preestablecida, 679  
acciones sobre registros referenciados, 679  
acelerado, 608, 692. *Vea también* DBMS  
  distribuidos; Escalamiento  
  arquitectura del servidor de medios especializados, 692  
ACID, propiedades, 518-519, 679  
ACID, transacciones, 542  
acoplamiento apretado, 616, 689  
acumulación de la tabla de hechos, 570, 679  
actividad de modificación, 465  
actividad preventiva, mejora de la calidad de los datos, 28  
actualización, costos fijos en contra de costos variables de, 595  
actualización de la pérdida de escolar, 537, 691. *Vea también* actualización pérdida  
actualización diferida, 529  
actualización pérdida, 520-521, 687. *Vea también* Actualización pérdida del escolar  
actualización de un almacén de datos, 555  
actualizaciones de vista, 348-349, 351-352  
actualizar operaciones en una vista  
  actualizable, 348-349  
administración de base de datos de objetos  
  arquitectura para, 649-655  
  razones empresariales para la, 642-643  
administración de bases de datos, herramientas de, 485-497  
administración de bloques, distribución entre sitios, 633  
administración de dependencias por parte de DBA, 494  
administración de la excepción, sentencias, sentencia ROLLBACK en, 517  
administración de la toma de decisiones, apoyo de la base de datos para la, 482-483  
administración de objetos distribuidos, 541  
administración de flujos de trabajo, 539-542  
administración de inventarios, 19  
administración del conocimiento, 484, 686  
administración de los recursos de la información, 19, 483-484, 686  
administración de procesos, 609, 612  
administrador de datos distribuidos (DDM), 620  
administrador del control de concurrencia, 523, 681, 689  
administradores de bases de datos. *Vea* DBA  
administradores de datos. *Vea* DA  
administradores heterogéneos de datos  
  locales, 621-622  
administradores locales de datos (LDM, por sus siglas en inglés), 620  
AFTER ROW, disparador  
  definición para mantener las columnas derivadas, 469  
  propagación de las actualizaciones, para, 408-409  
  reportes de excepción, para, 413-414  
agregados, cálculo a partir de agrupaciones múltiples, 313  
agregar cálculos, 323  
agregar enlace con una consulta de rescribir, 586  
agregar expresiones en la cláusula ORDER BY, 97  
agregar funciones, 92, 679, 693  
  con una función de agregar, 312  
  utilizada con un operador de resumen, 65-66  
agrupación  
  combinación con intersecciones, 110-111  
  efectos de los valores nulos en, 323-324  
  intersecciones, con, 94-95  
  proceso de evaluación y, 101  
aislamiento, niveles de, 536-537, 686  
álgebra, 56  
álgebra relacional, 56, 690  
  búsqueda automática más allá de las operaciones de, 377  
  consulta, 269  
  naturaleza matemática de, 57  
  operadores de, 56-67  
algoritmo de encriptación de llave pública, 487  
algoritmo de fusión aleatoria, 271, 692. *Vea también* intersección hash; loops anidados  
algoritmo de intersección de estrella, 271, 692. *Vea también* Índices de bitmap  
algoritmos de bifurcación, 271  
algoritmos de intersección, 270-271, 686  
almacenamiento digital para datos complejos, 642  
almacenamiento directo de datos  
  multidimensionales, 562  
almacenamiento estable, 4  
almacenamiento no volátil, 695  
almacenamiento volátil, 250, 695  
alternativas factibles, generación de, 32-33  
altura de un Btree, 261  
análisis de forma, 429-435  
análisis de sintaxis y semántica, durante la optimización de consultas, 268  
análisis inconsistente. *Vea* Resumen incorrecto  
analistas/programadores, 18, 679  
anidación en reportes jerárquicos, 359-360  
anomalías de modificación, 687  
  ejemplos de, 220  
  eliminación de, 225  
  evitar las, 220  
  resultado de las relaciones derivadas, 232  
anormalidades de actualización, 220  
ANSI (Instituto Nacional Estadounidense de Estándares, por sus siglas en inglés), 679  
API (Interfase de Programación de la Aplicación, por sus siglas en inglés), 650-651  
aplicaciones  
  especificación de la importancia relativa de, 464, 465  
  identificación de las bases de datos utilizadas por, 379  
  participación de una mezcla de datos simples y complejos, 643  
  vista de construcción de bloques, para, 353-354  
aplicaciones de negocios, requiere de grandes cantidades de datos, 642  
apoyo a la decisión  
  enlace con las realidades de los datos disponibles, 591  
  procesamiento de transacciones, en contra del, 554  
apoyo de las bases de datos en la toma de decisiones administrativas, 482-483  
árbol balanceado, 260  
archivos, 250, 683  
archivos cifrados, 257-259, 267, 268, 684  
archivos cifrados dinámicos, 259, 684  
archivos con función de montículo, 256  
archivos de árbol de múltiples vías. *Vea* Btree  
archivos secuenciales, 256-257, 259, 691  
archivos secuenciales desordenados, 256, 267, 268

archivos secuenciales ordenados, 256-257, 267, 268  
área global compartida (SGA, por sus siglas en inglés), 618  
arquitectura cliente-servidor, 16-17, 609-615, 680  
arquitectura de almacén de datos ascendentes, 557, 558, 680. *Vea también* Arquitectura de almacén de datos de tres capas, arquitectura de almacén de datos de dos capas  
arquitectura de almacén de datos de tres capas, 557, 693. *Vea también* Arquitectura de datos de raíz; Arquitectura de almacén de datos de dos capas  
arquitectura de arreglo disco.  
*Vea* Arquitectura de CD  
arquitectura de capas múltiples, 612-614, 687.  
*Vea también* Arquitectura de tres capas;  
Arquitectura de dos capas  
arquitectura de CN (nada agrupado), 616-617, 680  
arquitectura de componentes  
DBMS de objetos relacionales, para, 653  
procesamiento de bases de datos distribuidas, para el, 620-622  
arquitectura de disco compartida.  
*Vea* Arquitectura SD  
arquitectura de dos capas, 611-612, 694  
*Vea también* Arquitectura de múltiples capas; Arquitectura de tres capas;  
Arquitectura de almacén de datos de tres capas  
arquitectura de nada compartida.  
*Vea* Arquitectura de SN  
arquitectura de objetos grandes, 650, 655, 686  
arquitectura de SD (disco compartido, por sus siglas en inglés), 616, 691  
arquitectura de SE (todo compartido, por sus siglas en inglés), 616, 691  
arquitectura de servicios web, 614-615, 695  
arquitectura de SN (nada compartido, por sus siglas en inglés), 616, 692  
arquitectura de todo compartido.  
*Vea* Arquitectura SE  
arquitectura de tres capas, 612, 613, 693. *Vea también* Arquitectura de capas múltiples;  
Arquitectura de dos capas  
arquitectura de tres esquemas, 15, 16, 340, 693  
arquitectura orientada a tablas de tipos definidos por el usuario, 653  
arquitectura relacional de objetos, 655  
arquitecturas  
administración de bases de datos, para la, 649-655  
almacenes de datos, para, 556-558  
bases de datos de cliente-servidor, para, 609-615  
procesamiento paralelo de las bases de datos, para el, 616-617  
arquitecturas de esquema para el procesamiento de bases de datos distribuidas, 622-623  
arrastre dinámico, 380. *Vea también* Arrastres  
ARRAY, palabra clave, 663  
ARRAY, tipo de recolección, 657, 658  
arreglos, 653, 657, 658. *Vea también* Arreglo dinámico  
aserciones, 679  
en SQL:2003, 491-493  
sin el apoyo de bases de datos de importancia, 493  
asignación (=), símbolo, 381, 382  
asignación de peso y puntaje, 500-501  
asistente ER, 155, 180  
asociaciones  
almacenados por relaciones en ERD, 136-137  
en diagramas de clase, 157  
en UML, 158  
asterisco (\*) carácter comodín, 85, 86, 87-88  
ATM, transacción, 516, 517  
ATTRIBUTE, cláusula, 573  
atributo de cursor %IsOpen, 399  
atributo de cursor %Found, 399  
Atributo de cursor %RowCount, 399  
atributos, 679  
compuesto de separación, 173, 174, 178  
ERD, en, 136, 137

incorporación de historial a los, 175-176, 178  
incorporación de tipos de entidad, 431-432  
relaciones M-N, con, 142, 143  
tipos de entidad, de, 169  
transformación en tipos de entidad, 173, 174, 178  
atributos compuestos, separación, 173, 174, 178  
atributos de cursor con, 398  
declaración, 395  
*Vea también* Cursor; Cursor PL/SQL explícito  
auditoría, durante el mantenimiento del almacén, 592, 593  
autentificación, 386  
auto-intersecciones, 109-111, 691  
autorización  
derechos, 7  
reglas, 486, 679  
restrictiones, 488  
AVG, función, 92, 323

B

balanceo de carga, 617, 618, 686-687. *Vea también* Arquitectura CN (nada agrupado, por sus siglas en inglés); Arquitectura SN (nada compartido, por sus siglas en inglés)  
bandeja de auditoría, creación de disparadores, 403  
base de datos de captura de órdenes, 120-125  
base de datos del cliente para utilidades de aguas municipales, 171-173  
base de datos del servicio de agua, 5, 171-173  
base de datos de una universidad, 4-5  
CREATE TABLE, sentencias para, 73-74  
ERD para, 151  
base de datos de un hospital, 5-6  
bases de datos, 4, 682  
afinación, 7, 10-11  
artículos de acceso, 522  
características de, 4-6  
características esenciales de, 6  
componentes esenciales de los sistemas de información, como, 24  
contexto organizacional para la administración, 482-485  
definición, 6-8  
diseño, 429  
diseño de grandes, 31, 428  
distribución, 608  
encriptación de, 487  
identificación, 379  
interacción con, 17-18  
nivel de almacenamiento de las, 250-251  
planeación del desarrollo de nuevas, 19  
procesamiento del trabajo mediante procesamiento de lotes, 377  
propiedades de las, 4  
remotas, 629-630  
bases de datos de negocios, 6  
bases de datos homogéneas distribuidas, 628-629  
bases de datos locales en Oracle, 629  
bases de datos operacionales, 482, 688-689  
bases de datos relacionales  
diagramas, 140  
diseño, 220-223  
matemáticas de, 46  
tablas, 642-643  
terminología alternativa para, 49  
bases de datos relacionales de objetos, 653  
bases de datos remotas en Oracle, 629-630  
Bayer, Rudolph, 260  
BCNF (formato normal Boyce-Codd), 223, 227, 680  
relación con 3NF, 227-229  
violación como objetivo de diseño, 238  
violación de la partición de las tablas de Student Loan Limited, 463  
violaciones de, 227  
BEFORE ROW, disparadores, 411-413  
lineamientos, para, 495  
restricciones de integridad complejas, para, 406-408

BEGIN, palabra clave en un bloque, PL/SQL, 387  
bitácora. *Vea* Bitácora de transacción  
bitácora de transacción, 527, 592, 693  
bitácora de Web, 592  
bitácoras lógicas, 250  
almacenamiento en el orden de inserción, 256  
inserción en archivos secuenciales, 257  
bitácoras redundantes, 527  
BLOB (objeto grande binario, por sus siglas en inglés), tipo de datos, 650, 679  
*Vea también* arquitectura grande de objetos  
bloqueo de dos fases, 634  
bloqueos, 522-523, 687. *Vea también* Bloqueos exclusivos; Candados de intento; Candados compartidos  
bloques, *Vea* Bitácoras físicas  
bloques anónimos, 386-388  
BOT (comienzo de la transacción, por sus siglas en inglés), 525  
Boyce-Codd, formato normal, *Vea* BCNF (Formato Normal Boyce-Codd, por sus siglas en inglés)  
Btree, 257, 264, 267, 268, 274-275, 680  
archivos, 259-266, 680  
costo de las operaciones, 261, 264  
estructura de, 259-260  
índice, como, 265  
bucle, 306, 688. *Ver también* Intersección cifrada; Algoritmo de ordenamiento  
búfer, 250, 680  
búfer de aplicación, 250  
búsqueda secuencial en los archivos hash, 259  
búsquedas automáticas  
problemas que involucran inconsistencia en las, 521-522  
selección de índices para, 466

C

caída automática en Oracle RAC, 618  
cálculo orientado a objetos, 644-649  
cálculo tolerante a fallas, 608  
cálculos de resumen en informes jerárquicos, 361  
calidad de datos  
dimensiones o características de, 27-28  
preocupación para la administración de los recursos de información, como, 484  
problemas, 594  
campos de forma  
agrupación en tipos de entidad, 431  
asociación con los tipos de entidad, 431  
enlace con los tipos existentes de entidad, 442  
campos, multipropósito, 594  
candido exclusivo de intención compartida, 524  
candidos C. *Vea* Candados exclusivos  
candidos compartidos, 523, 691  
candidos de intento, 524, 686. *Vea también* Esquemas  
candidos exclusivos, 523, 683  
candidos predicados, 356  
candidos S (compartidos), 523, 691  
caracteres de enlace de patrones, 88  
característica compartida de las bases de datos, 691  
característica de despliegue automático de las herramientas CASE, 35  
características de partición de las bases de datos (DPF, por sus siglas en inglés), 619, 682  
característica de pegamiento  
herramientas CASE, de las, 35  
Visio Professional, en, 37  
características de objetos en los diagramas de clase, 157  
característica interrelacionada de las bases de datos, 4, 686  
cardinalidad de dependencia de existencia, 150  
cardinalidad de valores sencillos, 150  
cardinalidad máxima, clasificaciones para, 138  
cardinalidades, 137-138, 151, 152, 680  
apoyado por UML, 157  
clasificación de, 138-140  
especificaciones, 432  
incorrectas, 182

- representación de pata de cuello, 138  
reversas, 182  
cardinalidades de relación, 137-140, 169  
consistencia, 434  
variacições en la notación ERD para, 156-157  
cardinalidades incorrectas, 182  
cardinalidades mínimas  
clasificación para, 138-139  
errores en las, 182  
cardinalidades reversas, 182  
carta de revelación, 450-451, 453, 455-457  
CASCADE, opción de eliminación, 190  
CASCADE, palabra clave, 55  
CASE, herramientas, 34, 680  
comerciales, 36-39  
desarrollo de conversiones, 183  
funciones de, 35-36  
CASE, herramientas comerciales, 36-39  
CASE, sentencia, 348-385  
castigo al desempeño de vistas complejas, 340  
categoría de conexión para las reglas  
del diagrama, 155  
categoría de contenido para las reglas  
del diagrama, 155  
categoría de nombres para diagramas de reglas, 155  
categorización de nombres, 688  
CD (arreglo de discos, por sus siglas en inglés),  
arquitectura de, 616, 680  
CHAR, tipo de datos, 47, 381  
CHECK, cláusula  
CREATE DOMAIN sentencia, en la, 490  
CREATE TABLE sentencia, en la, 513  
Check Diagram, botón en el asistente ER, 155  
CHECK restricciones en la sentencia CREATE  
TABLE, 491  
Chen ERD, 156  
Chen, notación, 145  
CHILD OF, palabras clave, 572  
ciclo de relación, 690  
ciclo de vida de desarrollo del sistema, 25-26  
ciclo de vida del desarrollo de los sistemas  
tradicionales, 25-26  
ciclos en ERD, 182  
ciclos de vida. *Vea* Información de ciclos de vida  
ciclos de vida de la información, 483, 685  
clases, 157, 644, 645, 680  
clasificación  
cardinalidades, de, 138-140  
modelo relacional de entidades, en el, 147-150  
clasificación de funciones con una sentencia  
SELECT, 583  
clasificación, por medio de ORDER BY, 95  
cláusula que hace referencia en un disparador  
Oracle, 403  
cláusulas condicionales, 382-386  
clave en DKNF, 237  
clave externa, 50, 684  
índices de intersecciones de apoyo, 277  
no empleadas en una notación ERD, 140  
valores nulos y, 52  
clave primaria combinada, 51, 680  
clave primaria compuesta, 51, 680  
claves de candidatos múltiples, 227-229, 230-231  
CLI (interfase de nivel de llamada, por sus siglas en  
inglés), 378-379, 680  
cliente, 16, 606, 680  
clientes delegados, 607  
clientes gordos, 611  
clínicas dentales, apoyo a, 643  
CLOB (Objeto Grande de Caracteres, por sus  
siglas en inglés), 650, 680. *Vea también*  
Arquitectura grande de objetos.  
cluster, 616, 617, 680. *Vea también* Arquitectura de  
CD (arreglo de disco), Arquitectura de CN  
(nada agrupado)  
CODASYL, estándar, 12  
Codd, Ted, 80  
codificación orientada al evento, 377  
código de interfase de usuarios, 607  
código, desnormalización para combinar columnas  
de, 281-282  
código, modificación incremental del, 648-649  
código, prácticas de
- administradores de bases de datos, establecidas  
por los, 493  
consultas con desempeño pobre, para, 272-274  
código de interfase de usuarios, 607  
código de presentación, 610  
coherencia de caché, 617, 680. *Vea también*  
Arquitectura arreglos de disco, arquitectura  
de disco compartida  
colisión, 258, 680. *Vea también* Archivos cifrados  
colocación de datos, 252  
colocación de la transacción, 520, 694  
ColorPoint, tipo, 657  
columna, calificación del nombre de la, 688  
columna, resumen en el perfil de una tabla, 254  
columna de llaves, 224  
columna de tipo de registro, 662-663  
columnas  
actualización, de las, 114  
agrupación en dos, 94  
agrupación en una sencilla, 92  
colocación en una sencilla, 95-96  
correspondencia de posición de las, 64  
muestra de todas, 86  
renombradas, 342  
valores nulos, con, 320-321  
columnas calculadas en informes jerárquicos, 360  
columnas con nuevo nombre, definir una vista  
con las, 342  
columnas de intersección. *Vea* Columnas enlazadas  
columnas enlazadas, identificación para formas  
jerárquicas, 355-356  
columnas estables  
como buenas candidatas para índices  
de bitmap, 278  
requeridas para índices de bitmap, 266, 267  
columnas no llave, 224  
columnas representativas, desnormalización para  
combinar, 281-282  
columnas sin enlace, 688  
conservando el resultado, 61  
intersección externa de un lado con, 306, 336  
comentarios sobre PL/SQL, 381  
comercio electrónico, 613, 614  
comienzo con una transacción (BOT), 525  
COMMIT, cláusula, 516-517  
Comité de Lenguajes de Sistemas de Datos. *Vea*  
CODASYL, estándar  
compañías de seguros de autos, 643  
compatibilidad. *Vea* Compatibilidad de unión,  
Verificación de completado, 593  
compatibilidad de unión, 64-65, 694  
compatible con la unión  
hacer tablas, 111-113  
tablas, 300  
compensación del costo beneficio para la calidad en  
los datos, 28  
complejidad de la tarea, 540  
complejidad en una organización, 428  
completado/restricciones de consistencia, 596  
comprender, 435  
que usa formas, 436  
refinamiento, 232-236  
representación, 140, 158  
comunicación del interprocesador, 617  
compresión, acerca de las decisiones sobre el  
formato del bitácora de la, 282  
comunicación, mejora con la documentación,  
179-180  
comunicación, tiempos para consultas distribuidas,  
631-632  
conciencia del usuario acerca de vistas  
materializadas en comparación  
con las tradicionales, 584-585  
condición AND con un valor nulo, 322  
condiciones compuestas, efectos de los valores  
nulos en las, 321-322  
condiciones de grupo, 334, 684  
condiciones de registro, 334, 691  
condiciones simples, efectos de los valores nulos  
en, 320-321  
conectividad de base de datos abierta (ODBC), 379,  
611, 612, 680, 681  
conexión de bases de datos, 379  
conexión directa en ERD, 170  
conexión indirecta en ERD, 170  
conexiones entre tablas, 47-49  
confiabilidad  
arquitectura RAID, 284  
logro de una gran, 526  
preocupación con DBMS relacionados a  
objetos, 653  
conflictos de los candados, 523  
conformidad  
evaluación de los estándares SQL, 81  
niveles del estándar SQL: 2003, 655  
conformidad mejorada del estándar SQL:2003, 655  
conformidad mínima con el estándar  
de SQL:2003, 655  
CONNECT, cláusula, 379  
CONNECT, comando, 386  
CONNECT, función, 488  
conocimiento del entorno de, 252, 253  
Consejo de Procesamiento de la Transacción (TPC),  
por sus siglas en inglés), 502, 520  
constantes booleanas, 381  
constantes de fecha, no proporcionadas en  
PL/SQL, 381  
constantes, especificación de las, 113  
constelación, esquema de, 567, 681  
*Vea también* Tablas de dimensión; Tablas  
de hechos; Esquemas de copo de nieve;  
Esquema de estrella  
CONSTRAINT, palabra clave, 50  
construcción de modelado de datos  
especializados, 182  
construcción de tiempo en Oracle, 583  
constructor de tipo de registro, 659-660  
consulta de diferencias por medio de SELECT, 113  
consulta interna. *Vea* Consultas anidadas  
consulta traducida, 269  
consultas, 8, 690  
análisis para errores, 268  
escritura de formas jerárquicas, 356-359  
operaciones de enlace adicional, con, 273  
prácticas de codificación para, 272-274  
procesamiento con referencias  
de vista, 344-346  
resumen de aquellas que acceden a una  
base de datos, 255  
uso de las vistas, 342-344  
vistas complejas, con, 273  
consultas agrupadas recursivas, 312-315  
consultas almacenadas, en lugar de consultas  
anidadas, 332-333  
consultas anidadas, 303, 688  
eliminar el tipo II, 273  
en la cláusula FROM, 312-314  
múltiples niveles, 304  
múltiples sentencias de SELECT de Microsoft  
Access en vez de, 332-333  
tipo I, 303-305  
consultas anidadas de tipo I, 303-305, 694  
en otra consulta anidada de tipo I, 304  
IN, operador, 305-306  
tablas relacionales referenciadas en sentencias  
DELETE, 304-305  
consultas anidadas de tipo II, 308, 694  
con la función COUNT, 310  
evitar, 273  
NOT EXISTS, operador en, 310-312  
para problemas de diferencias, 308  
consultas estructuradas en árbol, 109-110  
consultores de almacenamiento de resumen, 590  
contexto incrustado para SQL, 81-82  
contexto organizacional para la administración de  
bases de datos, 482-485  
contexto propio de SQL, 81  
control de acceso de las bases de datos, 486  
control de acceso discrecional, 486-487, 682  
control de acceso obligatorio, 138, 687. *Vea también*  
Dependencia de existencia; Relación  
opcional  
control de concurrencia, 519-526  
centralizado contra distribuido, 633  
herramientas, 522-526  
objetivo de, 520

control de concurrencia distribuido, 633. *Vea también* Control de concurrencia  
 control de mensajes como una función de middleware, 611, 687  
 conversión de esquema para Student Loan Limited, 461-462  
 conversiones implícitas de tipo, 273  
 coordinación distribuida de un protocolo de compromiso de dos fases, 635  
 Corporación Internacional de Datos (IDC, por sus siglas en inglés), 13  
 correspondencia de posición de las columnas, 64  
 COST, función, 92  
 consulta anidada en la cláusula HAVING, con una, 315-317  
 consultas anidadas, en, 312, 313  
 DISTINCT, palabra clave interna, 313-314  
 problema de división con DISTINCT interna, 318, 319  
 valores nulos, con, 323  
 CPU, uso del, 251  
 CREATE ASSERTION, sentencias, 81, 492-493, 512, 679  
 CREATE DIMENSION, sentencia, 571-574  
 CREATE DOMAIN, sentencia, 490-512  
 CREATE/DROP, sentencias, 489  
 CREATE/DROP ROLE, sentencias, 487  
 CREATE INDEX, sentencia, 279-280  
 CREATE MATERIALIZED VIEW, sentencia, 583-584  
 CREATE, privilegio, 489  
 CREATE ROLE, sentencia, 511-512  
 CREATE TABLE, sentencia, 46  
 base de datos universitaria, para la, 73-74  
 CHECK, restricciones en, 491, 513  
 CONSTRAINT, cláusula, 50, 51  
 designación de una clave primaria, 50  
 especificación de los identificadores de objetos en Oracle 10g, 665-666  
 NOT NULL, restricciones, 52  
 Oracle 10g, en, 667  
 Oracle, 10g SQL, en, 76  
 SQL:2003, sintaxis, 74-76  
 Student Loan Limited, para, 474-477 cuadros de verificación en informes jerárquicos, 360  
 CREATE TRIGGER, sentencia, 81  
 CREATE TYPE, sentencia, 657, 667  
 CREATE USER, sentencia, 488  
 CREATE VIEW, sentencia, 81, 340-342  
 WITH CHECK OPTION, 349  
 SQL:2003, sintaxis para, 372CUBE, operador, 681  
 aumento de la cláusula GROUP BY, 574-578  
 combinación con ROLLUP y GROUPING SETS, 582  
 comparación con el operador ROLLUP, 579  
 extensión de una consulta de rescribir, 588-589  
 creencia de valores uniformes, 272, 694  
 Cuarto Formato Normal (4NF, por sus siglas en inglés), 223, 235, 684  
 cubo de datos, 562, 681  
 arquitectura de almacenamiento que manipula directamente, 589  
 ejemplos de, 563-564  
 grado de celdas vacías en, 563  
 resumen de operaciones utilizadas, 564-566  
 cubo de datos multidimensionales, 560-562  
 cuerpo de la tabla, 693  
 cuerpo de una tabla, 46  
 cuestiones de diseño  
     bases de datos cliente-servidor, de las, 609-611  
     dirigidas en varias arquitecturas, 611-615  
 cursor Already\_Open, excepción, 391  
 cursor, atributos, 398, 399  
 cursor PL/SQL explícito, 396-398, 683. *Vea también* Cursor PL/SQL implícito  
 cursor PL/SQL implícito, 395-396, 685  
 CURSOR, sentencia, 683  
 cursos, 380, 395-398, 681  
 cursos dinámicos, 395. *Vea también* Cursos

## D

DA (administradores de datos, por sus siglas en inglés), 19, 681  
 planeación de datos desarrollada por, 497-498  
 responsabilidades de, 484-485, 503, 504-505  
*data warehouses*, 13, 503-504, 554, 681  
 actualización, 594-596  
 aplicaciones de, 559-560  
 apoyo de DBMS para, 567-591  
 arquitecturas para, 556-558  
 características de, 554-556  
 costo para actualizar, 595  
 disponibilidad, 596  
 flujo de trabajo para mantenimiento, 592-594  
 madurez de la implementación, 559-560  
 mantenimiento, 591-596  
 materialización preferida en, 344  
 metodología ascendente para, 557, 558  
 modelo de madurez, 560, 681-682  
 procesamiento, 13  
 DATE, tipo de datos, 47, 381  
 datos  
     aseguramiento de la calidad de, 27-28  
     colocación física de, 31  
     conflicto, en, 594  
     definición del significado de, 27  
     operador de producto vectorial extendido que genera excesivos, 59  
     remotos, 620  
     representación multidimensional de, 560-566  
     sin tiempo, 595  
     ubicación de, 31  
 datos cambiantes que se pueden consultar, 592, 690. *Vea también* Datos cooperativos que cambian; Datos registrados que cambian; Datos de snapshot que cambian  
 datos complejos, 642, 643  
 datos de cambio, 680. *Vea también* Datos de cambio cooperativos; Datos de cambio registrados; Datos de cambio que se pueden consultar; Datos de cambio instantáneos; clasificación de, 592  
 sistemas fuente, de los, 591  
 datos de cambio cooperativos, 592, 681  
     *Vea también* Datos de cambio registrados; Datos de cambio que se pueden consultar; Datos de cambio instantáneos  
 datos de cambio registrados, 592, 687. *Vea también* Datos de cambio cooperativos; Datos de cambio que se pueden consultar; Datos de cambio de snapshot  
 datos de cambio snapshot, 592, 692. *Vea también* Datos de cambio compartidos; Datos de cambio de bicátor; Datos de cambio que se pueden consultar  
 datos de fuentes, *dumps* periódicos de, 592  
 datos de las series de tiempo, 569  
 datos del dado del operador del cubo, 565, 566, 682  
 datos derivados  
     decisiones de Student Loan Limited sobre, 466  
     decisiones sobre almacenamiento, 282-283  
     mantenimiento de los, 469  
 datos distribuidos, razones empresariales para los, 608-609  
 datos integrados en los almacenes de datos, 555  
 datos locales  
     DBMS distribuidos, en, 620  
     formatos, 623  
 datos no volátiles en almacenes de datos (*data warehouses*), 555  
 datos no volátiles en dispositivos de almacenamiento, 256  
 datos multidimensionales  
     extensiones a la cláusula GROUP BY para, 574-583  
     modelado de datos relacionales para, 567-571  
 datos persistentes, 689  
 datos relacionales  
     conversión a XML, 672  
     modelado, 567-571  
 datos remotos en DBMS distribuidos, 620  
 datos replicados, 633  
 datos sucios, 528, 593  
 DB2  
     CN, estilo de procesamiento en paralelo para, 619  
     SELECT, diferencias de sintaxis, 131  
     tipos definidos por el usuario, 652  
 DB2 Enterprise Server Edition, 619  
 DB2, tipos definidos por el usuario, 652  
 DBA (Administradores de las Bases de Datos, por sus siglas en inglés), 18, 19-20, 682  
     administración de la dependencia por los, 494  
     administración de los disparadores y procedimientos almacenados, 493-495  
     consulta de las tablas de catálogo, 496  
     desempeño de una selección y proceso de evaluación, 499-501  
     modificación de tablas de catálogo, 496  
     monitoreo de los indicadores claves del desempeño, 519  
     responsabilidades de, 484-485, 503, 504, 505  
     revisión de los planes de acceso, 271  
     uso de los resultados TCP, 502  
 DBA, función en Oracle 10g, 488  
 DBMS (Sistemas de Administración de Bases de Datos, por sus siglas en inglés), 6, 682. *Vea también* DBMS distribuidos  
     apoyo al procesamiento distribuido, 16  
     apoyo de la actualización de vistas de tablas múltiples, 349  
     arquitecturas de, 14-17  
     búferes, 250, 251, 253  
     características de, 6-11  
     comerciales, 30  
     conocimiento del ambiente específico para, 253  
     costos elevados de cambio, 499  
     desarrollo de aplicaciones proporcionado en, 9 en contra de los lenguajes de programación, 649  
     evaluación de los dos o tres candidatos superiores, 502  
     evaluación en base de prueba, 502  
     generaciones de, 12-13  
     herramientas para acceso ajeno al procedimiento, 8  
     limitaciones del uso de vista en las consultas, 343  
     organización interna de, 14  
     perfíles de tablas construidas por, 253-254  
     programación de las capacidades del lenguaje, 9-10  
     responsabilidades en arquitecturas cliente-servidor, 17  
     respuesta a las consultas, 8  
     selección y evaluación de los, 498-503  
     SELECT, diferencias de sintaxis entre los más importantes, 131  
     software de terceros para, 11  
     tipos de, 11  
 DBMS, comerciales, 30. *Vea también* DBM  
 DBMS de código abierto  
     productos, 14  
     software, 502  
 DBMS de cuarta generación, 12, 13  
 DBMS de escritorio, 11, 682. *Vea también* DBMS; DBMS incrustados; DBMS empresariales  
 DBMS de objetos, 505, 649  
 DBMS de primera generación, 12  
 DBMS de segunda generación, 12  
 DBMS de tercera generación, 12  
 DBMS distribuidos, 620-623, 682. *Vea también* DBMS  
     empresariales, 11, 13, 682, 683. *Ver también* DBMS, DBMS de escritorio, DBMS incrustados  
 DBMS incrustados, 11, 682, 683. *Vea también* DBMS de escritorio, DBMS empresariales  
 DBMS integradas fuertemente, 622  
 DBMS integrados desacoplados, 621-622  
 DBMS integrados y distribuidos desacoplados, 622-623  
 DBMS locales homogéneos, 620  
 DBMS orientados a objetos, 13, 654-655, 688

- DBMS\_Output, paquete, 399  
 DBMS paralelos, 616, 689. *Vea también Escalamiento; Aceleración*  
 DBMS relacionales, 12, 690  
     características que apoyan a los datos multidimensionales, 567-591  
     elementos básicos de, 46-49  
     tipos de datos en, 642  
 DBMS relacionales de objetos, 652-654, 688  
 DBWR. *Vea Proceso de escritura de bases de datos*  
 DDBMS. *Vea DBMS distribuidos*  
 DDM (Administrador de Datos Distribuidos, por sus siglas en inglés), 620  
 DECIMAL, tipo de datos, 47, 381  
 decisiones, sensibilidad de tiempo de, 595  
 declaración de variables, 381-382  
 DECLARE, palabra clave, 381, 387  
 DECLARE, sección, en la sentencia CURSOR, 396-398  
 declaraciones ancladas, 382  
 DEFAULT, palabra clave, 38  
 DEFERRABLE, palabra clave, 493  
 definición de bases de datos, 6-8  
     cambios, 14, 340  
     sentencias, 81  
 definición de lenguaje de objetos (ODL, por sus siglas en inglés), 644, 654  
 definición de relaciones, ventana, 8  
 definición de tablas, ventana, 7-8  
 definiciones de tabla, 658-661  
 Definition\_Schema, tablas de catálogo en, 495  
 DELETE, privilegio, 487  
 DELETE, sentencia, 114-115  
     costo de mantener los índices como resultado de, 276  
     ejecución de un disparador para cada, 405-406  
     otorgamiento de otro tipo de uso de una consulta anidada de Tipo 1, 304-305  
     SQL:2003 resumen de sintaxis, 131  
 DELETE ANY, privilegio, 489  
 DELETE CASCADE, acciones, disparadores y, 416  
 delta, 592  
 dependencia completamente funcional, 223  
 dependencia de existencia, 141, 683. *Vea también Relación obligatoria*  
     dependencia de existencia, realización de un tipo de entidad, 138-139  
     dependencia de identificación, 141-142, 151, 174, 685. *Vea también entidades débiles*  
         categoría para las reglas de diagrama, 155  
     complejidad de, 153  
     ejemplos de, 147  
     regla de cardinalidad, 152, 153, 154  
     regla que aplica a ERD de Student Loan Limited, 461  
     reglas de conversión, 184, 185-186  
     reglas, 153-154  
     símbolo de notación de pata de cuervo para, 150  
 dependencia sin compromiso, 521, 694.  
*Vea también Tipos de datos*  
     no convencionales de lectura sucia, almacenamiento y manipulación, 13  
 dependencias entre los objetos de las bases de datos, 493-494  
 dependencias funcionales. *Vea FD*  
 dependencias transitivas, 226, 694. *Vea también FD (dependencias funcionales)*  
 DEREF, función, 669  
 desarrollo cliente-servidor, procedimientos almacenados que permiten flexibilidad para, 388  
 desarrollo de bases de datos  
     fases de, 28-32  
     función de normalización en, 237  
     habilidades en, 32-34  
     herramientas de, 34-39  
     introducción a, 23-39  
     metas de, 26-28  
     proceso de, 28-34  
     verificación cruzada con desarrollo de aplicaciones, 32, 33  
 desarrollo de la aplicación, 7  
 de las tablas de Student Loan Limited, 464-465  
 en las bases de datos, 9  
 verificación cruzada con el desarrollo de la base de datos, 32, 33  
 DESC, palabra clave, 96  
 descomposición, 225  
 DESCRIBE, comando, 386  
 descripción de bases de datos, niveles de, 15  
 Descripción Universal, Integración del Descubrimiento (UDDI, por sus siglas en inglés), 614, 615  
 descubrimiento guiado por los datos, 559  
 deshacer, comando, comparado contra ROLLBACK, 517  
 desempeño. *Vea también Complejidades en el desempeño de la base de datos distribuida*  
     que afecta al procesamiento, 631  
     medición de la base de datos, 31  
 desempeño de la base de datos  
     expertos, 33  
     medida combinada de, 251  
     mejoría, 284-285  
 desempeño del precio de los intercambios en la selección de DBMS, 502  
 desenlace  
     fragmentos, de, 625  
     restrictión, 148, 149, 150, 682  
 desequilibrio del sistema de captura, 642-643  
 deslizamientos redundantes de discos independientes. *Vea RAID*  
 desnormalidades de eliminación, 220  
 desnormalización, 238, 252, 253, 280-282, 682  
     almacenes de datos, de, 555  
     decisiones de Student Loan Limited, sobre, 466  
     situaciones para, 280-282  
 detalle de grano fino para el almacén de datos, 567, 568  
 detalles históricos, añadidos a un modelos de datos, 175-178  
 determinante mínimo, 223  
 determinantes, 221, 682  
 DETERMINES, cláusulas, 573-574  
 diagonal (/) en SQL\*Plus, 386  
 diagrama de dependencia funcional, 221-222  
 diagrama de instancias, 139, 140  
 diagramas de relación de entidades. *Vea ERD*  
 diccionarios de datos, 35, 495, 681  
     creación y mantenimiento con herramientas CASE, 442  
     inclusión de justificación de diseño en, 180  
     manipulación de, 495-497  
     tipos de, 495  
     Visio Professional, en, 37  
 diccionario de datos corporativos, 442. *Vea también Diccionarios de datos*  
 diccionario de recursos de la información (IRD, por sus siglas en inglés), 495, 496, 686  
 diccionario global (GD, por sus siglas en inglés), 620  
 dimensiones, 562, 682  
     detalles sobre, 563  
     representación de, 571-574  
 diseñador de bases de datos como político, 27  
 diseñador de transacción, 533  
 diseño conceptual, división para proyectos grandes, 31-32  
 diseño de base de datos física  
     ambiente, 252, 253  
     como secuencia de los procesos de toma de decisiones, 253  
     del sistema de Student Loan Limited, 465  
     desnormalización, durante, 280-282  
     dificultades de, 253  
     entradas, 252-256  
     fase de desarrollo de la base de datos, 31  
     meta de, 251  
     objetivo de, 251  
     registro de formateo durante, 282-283  
     salidas, 252, 253  
     vista general de, 250-253  
 diseño de consultas, ventana Microsoft Access, en, 9, 60-61  
 operador de enlace de un solo lado, 63  
 diseño de tablas sencillas, 220  
 diseño de vista, 31  
     con formatos, 429-438  
     vista general de, 428-429  
 diseños de tabla, transformación de ERD en, 30  
 diseños normalizados, ventajas de los, 280  
 disparador/procedimientos de las bases de datos versus disparadores/procedimientos de las aplicaciones, 493  
 disparadores, 402, 694. *Vea también Disparadores encimados*  
     administración, 493-495  
     clasificación en SQL:2003, 403  
     comparados con aserciones, 493  
     contenido de los, 493  
     ejecución, 402  
     lineamientos para el control de la complejidad, 495  
     lineamientos para la ejecución, 416  
     motivación y clasificación de, 402-403  
 disparadores aplicables de una declaración de SQL, 414  
 disparadores de actualización, 403  
 disparadores de registro, 403  
 disparadores encimados, 415-416, 495, 689.  
*Vea también Procedimientos de ejecución del disparador; Disparadores*  
 disponibilidad, 608  
 dispositivos para el almacenamiento de datos, volatilidad de, 526  
 DISTINCT, palabra clave  
     consulta almacenada con la palabra clave SELECT, 332  
     dentro de la función COUNT, 319  
     eliminación de columnas duplicadas, 104  
     eliminación de duplicados, 96-97  
     funciones agregadas internas, 313-314  
     no se permite en las consultas actualizables 1-M, 350  
     no se permite en las vistas de una tabla sencilla actualizable, 346  
     restricción del cálculo a valores de únicos de registro, 92, 93  
 distribución uniforme, 254  
 división del procesamiento  
     afecta el diseño de una base de datos cliente-servidor, 610  
     en el diseño de una base de datos cliente-servidor, 609  
 DKNF (formato normal de claves de dominio, por sus siglas en inglés), 223, 237  
 documentación  
     ERD, de, 179-181  
     estándares, 493  
     función de las herramientas CASE, 35  
 documentación de diseño, 180-181  
 documentación de la información, reglas del negocio especificadas como, 151  
 documentación, incompleta, 179  
 documentación, inconsistente, 179  
 dominios, 490-491  
 DOUBLE PRECISION, tipo de datos, 47  
 DPF (característica de partición de la base de datos), 619, 682  
 DROP ASSERTION, cláusula, 512  
 DROP DOMAIN, cláusula, 512  
 DROP, privilegio, 489  
 DROP ROLE, cláusula, 487, 511-512  
 DROP TABLE, cláusula, 76  
 DROP VIEW, cláusula, 372  
 Dup\_Val\_On\_Index, excepción de, 391

## E

- economías de escala, proporcionadas por el procesamiento de lotes, 377  
 EDM (modelo empresarial de datos, por sus siglas en inglés), 485, 556, 683  
 efectos colaterales de actualizaciones de vistas, 348  
 ejecución de consultas paralelas bajo máquinas ROLAP, 590

- ejecución del código máquina de planes de acceso, 271  
 ejecución recursiva, 415, 416  
 ejecución secuencial de transacciones, 520  
 ejecución simultánea de las transacciones, 520  
 eliminación de una estructura Btree, 261, 263  
 encabezado de la tabla, 46, 693  
 encapsulado, 645, 683  
 enciclopedia. *Ver* Diccionarios de datos  
 encriptación, 487, 683  
 enlace con los tipos de entidad existentes, 442  
 enlace de agrupación para una consulta de describir, 586  
 enlace de condición de registros para la escritura de una consulta, 586  
 enlace de índices, 265-266  
 enlace de subseries, descubrimiento de un problema de división, 316  
 enlace exacta, 88. *Vea también* Enlace inexacta  
 enlace exacta de cuerda, 683  
 enlace inexacta, 87-88, 685  
 entidad asociativa, tipos de, 144-146, 232, 679  
 entidades, 683  
 clasificación, 147  
 base de datos de hospitales, en una, 6  
 base de datos de una universidad, en una, 5  
 base de datos de un proveedor de servicios de agua, 5  
 ERD, en, 136  
 bases de datos, en, 4  
 entidades débiles, 141, 156, 695. *Vea también* Dependencia de identificación; Identificación de regla de relación notación del símbolo de pata de cuervo para, 150  
 transformación en fuertes, 174-175, 178, 186, 187  
 entidades fuertes, transformación de las débiles en, 174-175, 178  
 entornos de bases de datos, administración, 503-505  
 entornos distribuidos, 504-505  
 entradas del diseño físico de la base de datos, 252, 253-256  
 EOT (final de la transacción, por sus siglas en inglés), 525  
 equivalencia de relación, 146-147, 690. *Vea también* Tipos de entidad asociativa; Identificación de la regla de relación  
 ER/Studio 6.6, 36  
 ERD (diagramas de relación de entidades), 11, 29-30  
 añadir historia a, 175-177, 178  
 consistente con los problemas de las sentencias de narrativa, 168  
 conversión a diseños de tablas, 35  
 conversión a tablas relacionales, 183-195  
 creación para representar formas, 430  
 diseños más sencillos en contra de complejos, 168-169  
 documentación, 179-181  
 elementos básicos, 136-137  
 finalización, 179-183  
 introducción a, 136-140  
 mejoría, 151  
 para bases de datos grandes, 428  
 para el formato de factura, 435, 436  
 para el Loan Origination Form, 455, 456  
 refinamiento para Student Loan Limited, 461-464  
 refinamientos para, 173-178  
 reglas básicas de conversión para, 183-187  
 representación de las reglas de negocios en, 151  
 revelación de conflictos en, 35  
 revisión de la consistencia y completado, 433-434  
 ERD, esquema de estrella, 567, 568  
 ERD, notación  
 clasificación de apoyo, 147  
 comparada con otras notaciones, 156-159  
 restricciones en, 157  
 resumen de, 150-152  
 variaciones en, 156-157  
 error de tiempo de ejecución, 416  
 errores de diseño, detección de los más comunes, 181-183  
 errores de semántica, 268  
 errores de sintaxis, 268  
 errores, manejo inesperado, 392  
 escalabilidad  
 metodología de cliente-servidor, de la, 607  
 proporcionada por la opción DPF, 619  
 escalabilidad horizontal, 607  
 escalabilidad vertical, 607  
 escalamiento, 608, 691  
 escalamiento lineal, 608  
 escritura de operaciones a un almacenamiento no volátil, 528  
 escritura forzada, 528, 684  
 esfuerzos estandarizados para SQL, 80-81  
 espacio en disco, minimización, 251, 252  
 esparcir, 563, 692. *Vea también* Especialización de cubo de datos y administración de bases de datos, 485  
 especialistas de datos, 503, 504-505  
 especialistas en bases de datos, 497-503  
 especialización de funciones en el desarrollo de una base de datos, 33  
 especialización de la tarea para DA y DBA, 485  
 especialización del ambiente para DA y DBA, 485  
 especificaciones de diseño, 25  
 espejeo de discos, 690  
 esquema conceptual, 15, 16, 29, 461-464, 681.  
*Vea también* Vista externa; Esquema interno; Esquemas; Arquitectura de tres esquemas  
 esquema conceptual global, 623  
 esquema de estrella, 567, 692. *Vea también* Esquema de constelaciones; Esquemas de copo de nieve; Tablas de dimensión; Tablas de hechos;  
 representación de tiempo en, 569-571  
 uso repetido para relaciones M-N, 568-569  
 variaciones a, 567-569  
 esquema de fragmentación, 622  
 esquema de información, 495  
 esquema de ubicación, 622  
 esquemas, 15, 691. *Vea también* Esquema conceptual; Vista externa; Esquema interno; Arquitectura de tres esquemas definición, 29  
 Oracle, en, 489  
 esquemas de copo de nieve, 568, 570, 590, 692. *Vea también* Esquema de constelación; Tablas de dimensión; Tablas de hechos; Esquema de estrella  
 esquemas de mapeo local, 623  
 esquemas externos, 29  
 estampas de tiempo, 555  
 estándar de encriptación de datos, 487  
 estándar SQL:1999, 74, 81, 312  
 estilo del operador de intersección, 106-109, 298, 686  
 combinación con el estilo del producto vectorial, 108  
 combinación de una consulta anidada de Tipo I con, 303-304  
 comparación con el estilo del producto vectorial, 109  
 DELETE, eliminación de la sentencia, 115  
 mezcla de intersecciones internos y externos, 301  
 estilo tabular, 9  
 estilos de definición de una tabla, 660  
 estilos de lenguaje para integrar un lenguaje de procedimiento, 378-379  
 estrategia de dividir y conquistar, 31, 428  
 estrategia de materialización para procesar consultas que hacen referencia a vistas, 344  
 estrategia de modificación para procesar consultas que hacen referencia a las vistas, 344-345  
 estructura Btree dinámica, 260. *Vea también* Btree  
 estructura Btree orientada a bloques, 260  
 estructura Btree ubicua, 260  
 estructura compacta Btree, 260  
 estructura de la forma, 430-431, 684  
 estructura de la tarea, 540  
 estructura jerárquica  
 definición para una forma, 430-431  
 para el formato de factura, 435  
 estructura primaria de archivo, 265, 267, 268, 689.  
*Vea también* Estructura secundaria de archivo  
 estructura secundaria de archivo, 265, 267-268, 691.  
*Vea también* Estructura primaria de archivo  
 estructuras de archivos, 252, 253  
 características de, 267-268  
 disponibles en la mayoría de las DBMS, 256-268  
 estructuras de dependencia complejas, 231  
 estructuras de dependencia simple, 231  
 estudio de factibilidad, 25  
 ETL, herramientas, 594, 683  
 eventos, 610. *Vea también* Evento aplicable; Eventos del disparador  
 eventos aplicables, clasificación de disparadores por, 403  
 eventos de disparador, combinación, 409  
 EXCEPT, palabra clave, 308  
 DB2, en, 131  
 SQL:2003, en, 113, 306  
 excepción, administración durante auditoría, 593  
 excepción definida por el usuario, 390  
 excepción de Invalid\_Cursor, 391  
 excepción No\_Data\_Found  
 (`no_se_encontr_información`), 391  
 excepción Rowtype\_Mismatch, 391  
 excepciones en PL/SQL, 390, 391  
 excepciones predefinidas, 390, 391  
 exactitud de mayúsculas/minúsculas, empate exacto e inexacto y, 88  
 EXECUTE, comando, 386  
 EXECUTE, privilegio, 487  
 EXISTS, operador, 308  
 EXIT, sentencia, 385  
 expresiones, 82, 683  
 expresiones de camino en una sentencia  
 SELECT, 664  
 expresiones lógicas, 687  
 complejas, 89  
 eliminación de las partes redundantes de, 268-269  
 usadas en el operador de restricción, 57  
 extracción en el mantenimiento del almacén de datos, 592, 593
- F
- Fagin, Ronald, 237  
 falla en el sistema operativo, 527  
 fallas  
 detección en el procesamiento de compromiso distribuido, 633-634  
 tipos de, 526  
 fallas detectadas por el programa, 526  
 fallas en el sistema, 526, 527, 529, 550  
 fallas locales, recuperación de, 529  
 fallo del dispositivo, 526, 527, 529  
 familias de subtablas, 653, 661-664  
 fase creciente de 2PL, 525  
 fase de actualización en el mantenimiento del almacén de datos, 593  
 fase de decisión en un proceso de compromiso de dos fases, 634, 635  
 fase de diseño de la base de datos distribuidos del desarrollo de la  
 procesamiento de bases de datos distribuidos, 624-630, 631-635  
 fase de diseño para base de datos lógica en el desarrollo de la base de datos, 30  
 fase de mantenimiento del ciclo de vida del desarrollo de los sistemas, 26  
 fase de preparación del mantenimiento del almacén de datos, 592, 593  
 fase de votos en un procesamiento de compromiso de dos fases, 634-635  
 fase preliminar de investigación del ciclo vida del desarrollo de los sistemas, 25

FD (dependencias funcionales, por sus siglas en inglés), 221-223, 684  
 agrupadas a partir de los campos dentro de los tipos de entidad, 431  
 agrupadas por determinantes, 230  
 derivadas de otras FD, 229  
 derivadas de una consulta de describir, 586  
 eliminación del potencial, 223  
 escritura con un lado nulo de  
   mano derecha, 238  
 identificación, 222-223  
 listado, 222  
 MVD como generalizaciones de, 235  
 para las tablas iniciales de Student Loan Limited, 462-463  
 para relaciones 1-M, 222  
 reglas acerca de, 223  
**FETCH**, sentencia, 396  
**fijación**, 679. *Vea también* Planes de acceso  
 mensajes para un lenguaje de programación de una base de datos, 379  
 mensajes para una implementación de un método, 648  
 reducción en la velocidad de ejecución para consultas complejas, 273, 274  
**fijación de estándares para bases de datos corporativas**, 442  
**fijación dinámica**, 379, 648, 679. *Vea también*  
 Fijación  
**FileMaker Pro**, 14  
**Firebird**, 14  
 flexibilidad en la tecnología de cliente-servidor, 607  
**FLOAT**, tipo de datos, 47, 381  
 flujos de clic. *Vea también* Historial de acceso  
 flujos de trabajo, 540, 695  
   afectados por la división  
     del procesamiento, 610  
   caracterización, 540  
   clasificación, 540, 541  
   especificación e implementación, 541-542  
   flujos de trabajo orientados a los humanos, 540  
   flujos de trabajo orientados por computadora, 540  
**FOR EACH ROW**, palabras clave, 403  
**FOR LOOP**, sentencia, 385  
 forma principal, 353, 354, 687  
   consulta para, 357, 359  
 formas, 353, 684  
   entre relación de precedencia, 441  
   implementación, 467  
   para introducción de datos, 9, 10, 353  
   resumen de aquellos que accesan  
     a la base de datos, 255  
   uso de relaciones M-forma, 435-438  
   ver diseño con, 429-438  
 formas jerárquicas, 353-354, 684  
   bosquejo de una opción de barrera de la transacción, 534-535  
   escritura de consultas para, 356-359  
   habilidades de formación de consultas para, 355-359  
   relación con tablas, 354-355  
   vistas en, 353-359  
 formas normales, 223-232, 688  
   nivel superior, 236-237  
   relación de, 223-224  
 formateo de datos, 252  
 formateo de registro, decisiones, 282-283  
 formato centralizado de compra, 437-438  
 formato de compra de proyecto, 436-437  
 formato de factura, 434-435, 436  
 formato de legado, datos fuente en un, 591  
 formatos de entrada de datos. *Vea* Formatos  
 formato del curso de la bitácora, 438  
 formulación de la consulta  
   formas jerárquicas, para, 355-359  
   informes jerárquicos, para, 361  
   preguntas críticas para, 101-103  
   refinamiento de ejemplos, 103-113  
   SQL avanzado, 279-324  
   SQL, con, 79-116  
 fórmulas de costo, evaluación de los planes de acceso, 271  
**FOR**, sentencia, cursor interno implícito, 395

**FoxPro**, 14  
 fragmentación en el almacenamiento de la base de datos, 285  
 fragmento horizontal derivado, 625  
 fragmento mixto, 622  
 fragmentos, 684. *Vea también* Operador de semi-intersecciones  
 DBMS distribuidos e integrados  
   estrechamente, en, 622  
   definición, 624-625  
   ubicación en sitios, 626  
 fragmentos horizontales, 625  
 fragmentos verticales, 625  
 frecuencia de refresco, 595  
**FROM**, cláusula  
   consultas anidadas en, 312-314  
   operaciones de intersección, en, 91  
   sintaxis para consultas anidadas en, 333  
 fronteras de transacción, 533-535, 693  
 fuentes de datos  
   determinación de la frecuencia de actualización para, 595  
   disponibles para poblar el almacén de datos, 591-592  
**FULL JOIN**, palabra clave, 300  
 función de diagramación de las herramientas CASE, 35  
 función más baja, 88  
 función superior, 88  
 funciones, 221  
   asignación a usuarios, 488  
   creación en SQL:2003, 487  
   PL/SQL, en, 392-394  
   SQL:2003, en, 657  
 funciones cifradas, 258  
 funciones de análisis para las herramientas CASE, 35  
 funciones de la base de datos, clasificación de, 17  
 funciones de radio con una sentencias SELECT, 583

**G**

**GCS** (Servicio Global de Caché, por sus siglas en inglés), 618  
**GD** (diccionario global, por sus siglas en inglés), 620  
 generalización, construida en el UML, 158  
 generalizar. *Vea* Operador de cubos que generaliza  
**GENERATE**, cláusula, 76  
**GRANT/REVOKE**, sentencias, 487-488, 489  
**GRANT**, sentencia, 81  
 granularidad de bloqueos, 523-524, 687  
 granularidad  
   candido, 523-524  
   clasificación de los disparadores por, 403  
**GROUP BY**, cláusula, 99-100, 102-103  
   comparación con ORDER BY, en, 96  
   efectos de los valores nulos en, 323-324  
   extensiones para los datos multidimensionales, 574-583  
   operador CUBE como extensión de, 681  
   que ocurre conceptualmente después de WHERE, 101  
**GROUP BY**, palabra clave  
   no permitida en consultas 1-M que se pueden actualizar, 350  
   no permitida en vistas de tabla sencilla que se pueden actualizar, 346  
**GROUPING SETS**, operador, 580-582, 684  
 Grupo de Administración de Bases de Datos de Objetos (ODMG, por sus siglas en inglés), 654  
 grupo de aplicaciones reales en Oracle. *Vea* RAC  
 grupos de Reales de Aplicación. *Vea* RAC  
 grupos en informes jerárquicos, 360  
 grupos que se repiten, desnormalización, 280, 281  
**GSL** (Guaranteed Student Loan), programa, 450  
 guion bajo \_, que enlaza a cualquier carácter sencillo, 88

**H**

habilidades cualitativas, 32  
 habilidades difíciles, 33, 34  
 habilidades suaves, 32, 34  
**HAVING**, cláusula, 100  
   comparada con WHERE, 93  
   COUNT, función con consulta anidada en, 315-317  
   eliminación de las condiciones que no contienen funciones junto a la cláusula GROUP BY, 103  
**HELP**, comando, 386  
 herencia, 148, 645-647, 686  
   aplicación a tablas, 661  
   extensión a niveles múltiples, 647  
   apoyada para tipos definidos por el usuario, 665  
 herramientas CASE front-end, 34  
 herramientas de análisis en Visio, 38-39  
 herramientas de Extracción, Transformación y Carga (ETL, por sus siglas en inglés), 594, 683  
 herramientas de ingeniería de software por medio de computadoras. *Vea* CASE, herramientas  
 herramientas de visualización de datos, 559  
 herramientas gráficas para tener acceso a las bases de datos, 8-9  
 herramientas por medio de computadoras, con apoyo de la selección de índices, 277  
 hija, tabla, 350, 351  
 hijos. *Vea* subtipos  
 hipercubo. *Vea* Cubo de datos  
 histograma de alturas iguales, 254-255, 272, 684  
 histograma de anchos iguales, 254, 272, 685  
 histogramas, 254, 684-685  
 historia limitada, añadirla a un tipo de entidad, 177  
 historial de acceso, 592  
 hoja de datos, 682  
**HOLAP** (OLAP híbrido), 590-591, 685  
 homónimos, 441, 442, 443-444, 685. *Vea también* Sinónimos  
 Hot spots, 520, 535, 685  
 Hot spots dependientes del sistema, 534, 685  
 Hot spots independientes del sistema, 534, 685  
**HTML** (Lenguaje de Señalización de Hipertexto, por sus siglas en inglés), 613, 685  
 Hubs en ERD, 170

**I**

**IBM**  
 CODASYL, estándar ignorado por, 12  
**IDC** (Corporación Internacional de Datos, por sus siglas en inglés), 13  
 identificadores de objetos, 644  
   generadas por el usuario, 663  
   que especifican en Oracle 10g, 665-666  
 identificadores de objetos generados por el usuario, 663  
 identificadores de usuarios, 380  
 identificadores múltiples, 594  
**IF**, sentencia, 382, 383-384  
**IF-THEN**, sentencia, 383  
**IF-THEN-ELSE**, sentencia, 383, 384  
**IF-THEN-ELSIF**, sentencia, 383, 384  
 imágenes, almacenamiento digital de, 642  
 Immon, William, 554  
 implementación  
   de un método, 644  
   decisiones para Student Loan Limited, 467  
   eficaz, 28, 31  
 implementación efectiva, 28, 31  
 inclusión de serie, 662, 668  
**IN**, operador, 93  
 independencia, 232  
 independencia de datos, 15, 340, 681  
   aseguramiento de, 16  
   proporcionada por encapsulado, 645  
 independencia de relación, 232-234, 690  
 índice de bitmap de intersección, 266-267

- índices, 31, 274, 685. *Vea también* Índices de bitmap; Btree; Archivos cifrados  
evitar combinaciones con columnas, 278  
selección para Student Loan Limited, 465-466  
índices agrupados, 274-275, 680, 685.  
*Vea también* Índices no agrupados;  
Estructura primaria de archivo  
Índices no agrupados, en comparación con, 276-277  
mantenimiento caro, de, 277  
índices compuestos, reglas de enlace, 265  
índices de bitmaps, 266-267, 268, 679. *Vea también*  
Algoritmo estrella de intersección  
índices de columnas sencillas, enlace  
con las reglas, 265  
índices de intersección de bitmaps, 266, 590  
índices sin grupo, 274, 275, 685, 688.  
*Vea* Índices de grupo; Índices  
selección; Estructura secundaria de  
archivos  
columnas con muchos valores como una buena  
opción para, 277  
combinar varias columnas y, 278  
comparaciones para agrupar, 276  
en una llave foránea, 277  
industrias, proyectos de almacén de datos  
en las, 559  
Informe de Actividad de Préstamo, requisitos  
de datos para, 468-469  
informes, 690. *Vea también* Informes jerárquicos  
implementación, 467  
proporcionados por las bases de datos, 9, 10  
resumen de aquellos que tienen acceso a la base  
de datos, 255  
vistas en, 359-362  
informes de actividad de préstamos  
de Student Loan Limited, 454  
modelado de datos de, 459-460  
informes de falla en el control. *Vea* Conversión de  
informes jerárquicos  
fase de diseño de una base de datos  
lógica, en la, 30  
producción de un diseño de tabla para ERD, 35  
volúmenes de procesamiento como un  
impedimento para, 467  
informes jerárquicos, 359-362, 684  
ingeniería hacia delante, 35, 684. *Vea también*  
Herramientas CASE;  
ingeniería reversa, 35, 391. *Vea también*  
Herramientas CASE; Ingeniería  
hacia delante  
Ingres, 14  
INNER JOIN, operación, 305  
INNER JOIN, palabras clave, 651  
inserción de anomalías, 220  
inserciones a la estructura Btree, 261  
INSERT ANY, privilegio, 489  
INSERT, sentencia, 113-114  
costo de mantener índices, 276  
resumen de sintaxis de SQL:2003, 130  
INSERT, privilegio, 487  
insertar operaciones  
impacto de las registros referenciados en, 55  
vistas que se pueden actualizar, en las, 348  
INSTANTIABLE, palabra clave, 657  
INSTEAD OF, disparador, 403  
INTEGER, tipo de datos, 47, 381  
integración, 482, 483  
estrategia, 440, 686  
fase de mantenimiento del almacén  
de datos, 593  
restricciones, 595-596  
integración de la vista, 439-444  
ejemplos de, 442-444  
metodologías para, 439-442  
proceso, 31, 429  
integración incremental, 455-460, 685, 686  
integridad de exclusividad. *Vea* Integridad de  
entidades  
integridad de los tipos de candidatos, 169  
integridad de una entidad, 49, 50, 683  
integridad histórica, preserva para las tablas de  
dimensión, 570-571  
integridad referencial, 49, 50-51, 690. *Vea también*  
Llaves candidatas; llaves extranjeras;  
Llaves primarias  
regla, 50  
restricciones, 51, 225, 230, 666, 667  
representación gráfica de, 53-54  
para relaciones auto-referenciadas  
(unaria), 52-53  
interacciones del disparador, 494-495  
interfase de nivel de llamada (CLI, por sus siglas en  
inglés), 378-379, 680  
interfase de programación de la aplicación (API),  
650-651  
interfase del paquete, 399  
integración paralela, 686, 689  
Internet, 686  
DBMS y, 13  
como el área con más estándares, 607  
estándares para los servicios Web, 614, 615  
procesamiento distribuido en, 16  
interoperabilidad, 607, 610-611  
interpretación de los planes de acceso, 271  
intersección cifrada, 271, 684. *Vea también*  
Algoritmo de fusión aleatoria  
intersección-equí, operador, 683. *Vea también*  
Operador de intersección; Operador de  
intersección natural  
intersección externa completa, 300-301, 336  
intersección híbrida, 271, 685. *Vea también* Bucles  
anidados; Algoritmo de fusión aleatorio  
intersección, operador, 63, 64, 65, 686  
intersecciones  
agrupación con, 94-95  
combinación de tablas con, 102  
problemas de división con, 316-317  
que se agrupan, combinación con, 110-111  
que domina el mercado del software de bases  
de datos de, 115  
intersecciones de inequidad, 307  
intersecciones externas  
mezcla con las internas, 301-302  
notaciones en Oracle 8i, 335-337  
intersecciones internas, 301-302, 336  
INTERSECT, consulta, 112  
Intranet, 686  
IRD. *Vea* Diccionario de Recursos  
de Información  
ISA, relación, 148  
IS NOT NULL, operador de comparaciones, 89  
IS NULL, condición, 308  
IS NULL, operador de comparaciones, 89, 306  
ISO (Organización Internacional de Estándares, por  
sus siglas en inglés), 496, 686
- J
- JDBC (Conectividad de Bases de  
Datos en Java, por sus siglas en inglés),  
379, 611, 680, 681  
jerarquía de dimensiones, 682  
jerarquía de organización, localización de todos los  
subordinados en, 110  
jerarquía explotada, 563  
jerarquías  
dimensiones, de, 563  
especifican en una sentencia de CREATE  
DIMENSION, 572  
jerarquías de generalización, 148, 151, 152, 684.  
*Vea también* Subtipos; Súertipo  
adicción, 177-178  
conversión, 190-191  
desnormalización, 280-281  
niveles múltiples de, 149-150  
propiedades, para, 659  
regla de conversión, 190-191  
regla de participación, 152, 153  
símbolo de notación de Pata  
de Cuervo, 150  
uso excesivo de, 182  
JOIN KER, cláusula, 572, 573  
llaves candidatas con propósito simple, 169  
llaves primarias estables, 169  
lógica de negocios, 610  
lógica de validación, 610  
LONG, tipo de datos, 47  
LOOP, sentencia, 385, 386
- L
- la fuerza detrás de SQL, 80  
LDM (administradores locales de datos, por sus  
siglas en inglés), 620  
lectura no repetible, 522, 688  
lectura no trivial MVD, 235, 684, 688  
LEFT JOIN, palabra clave, 299  
lenguaje de bloques estructurado,  
como PL/SQL, 386  
Lenguaje de Consultas de Objetos (OQL, por sus  
siglas en inglés), 654  
integración de consultas, 273, 274, 690  
intersecciones de tabla, 107-109  
Lenguaje de Consultas Estructurado. *Vea* SQL  
Lenguaje de Descripción de Servicios Web (WSDL,  
por sus siglas en inglés), 614, 615, 695  
Lenguaje de Flujo de servicios Web (WSFL, por sus  
siglas en inglés), 614, 615  
Lenguaje de Modelo Unificado. *Vea* UML  
Lenguaje de Módulos Persistentes  
Almacenados, 378  
lenguaje de procedimiento  
interfase, 7, 9-10, 689  
interrogación con un lenguaje ajeno al  
procedimiento, 378  
Lenguaje de Programación/Lenguaje de Consultas  
Estructuradas. *Vea* PL/SQL  
lenguaje formal de reglas, 151  
lenguaje natural correspondiente al Diagrama  
Entidad Relación (DEF), 137  
lenguajes de programación  
añadir capacidades completas de, 9-10  
contra DBMS, 649  
generaciones de, 12  
orientados a objetos, 649  
tipos de datos de, 642  
lenguajes de programación de las bases de datos,  
376, 682. *Vea también* CLI (Interfase de  
Nivel de Llamada, por sus siglas en inglés)  
cuestiones de diseño que involucran a los,  
378-380  
ligamiento, 379  
motivaciones para, 376-378  
LGWR, (proceso de escritura de bitácora, por sus  
siglas en inglés), 618  
LHS (lado izquierdo, por sus siglas en inglés),  
221, 223, 682  
ligamiento tardío, 648  
ligas de bases de datos, 629-630, 682  
limpieza en el mantenimiento de un almacén de  
datos, 592, 593  
lineamientos de análisis para problemas de  
narrativa, 170  
líneas de detalle, 360, 682  
LIKE, operador, 87  
listas de parámetros en funciones, 392  
llaves candidatas, 50, 151, 169, 224, 680  
declaración con la palabra  
clave UNIQUE, 50-51  
dependencias funcionales que identifican el  
potencial, 221  
múltiples, 227-229, 230-231  
llaves nodo, 431, 688  
llaves primarias, 30, 50, 689  
como buenas candidatas para la agrupación de  
índices, 277  
determinación de columnas ajenas  
a las llaves, 225  
determinación, 169, 170  
generación de valores únicos para, 76-77  
identificador de la base de datos de servicios de  
agua, 171-172  
préstamo de otros tipos de entidades, 141  
que dan apoyo a la identificación de entidades,  
151, 152  
tipos de entidades, de, 136  
llaves primarias con propósito simple, 169  
llaves primarias estables, 169  
lógica de negocios, 610  
lógica de validación, 610  
LONG, tipo de datos, 47  
LOOP, sentencia, 385, 386

LR. *Vea* Bitácoras lógicas  
 LSN (número de secuencia de bitácora, por sus siglas en inglés), 527  
 tabla M (hija) en Microsoft Access, 350

## M

macros, comparadas con las vistas, 340  
 Malhotra, Yogesh, 484  
 mantenimiento de dependencia de firma, 494  
 mantenimiento de dependencia *timestamp*, 569  
 mapeos de esquema, 15-16, 691  
 matemáticas para bases de datos relacionales, 46  
 materialización de la vista, 344, 694  
 MAX, función, 92  
 medida combinada del desempeño, 255  
 medidas  
   celdas de cubos de datos, en, 562, 563  
   detalles acerca de las, 563  
   que derivan, 563  
 medidas derivadas, 563  
 memoria principal, 250, 251, 252  
 mensajes, 648, 687. *Vea también* Fijación  
 mercado DBMS incrustados, 14  
 mercado de datos a tiempo, 557  
 mercado de software de bases de datos de escritorio, 14  
 mercado meta, aplicación de las técnicas de minería de datos, 558  
 mercados de datos, 577, 681  
 mercados operacionales, 557, 689. *Vea también* Mercados de datos  
 metadatos, 495, 681, 685. *Vea también* Diccionarios de datos  
 método del constructor, 657  
 métodos  
   implementaciones múltiples de, 647  
   objetos, en, 644  
 métodos de mutación, 657  
 métodos observadores, 657  
 métodos propietarios en contra de los estándares abiertos, 607  
 metodología, 14  
 metodología de actualización inmediata, 529-530, 531, 532, 685. *Vea también* Metodología de actualización diferida; Bitácora de protocolo  
 metodología de actualizaciones diferidas, 530-532, 682  
 metodología de diseño inicial para técnicas de normalización, 237-238  
 metodología de islas-de-automatización, 497  
 metodología de refinamiento, 237-238  
 metodología incremental para ver la integración, 439, 442-443  
 metodología paralela para la integración de la vista, 439-440, 443-444  
 metodología pesimista para el control de concurrencia, 525  
 metodologías de desarrollo, 26  
 metodologías de desarrollo de aplicación rápida, 26  
 metodologías de desarrollo en espiral, 26  
 M-forma (multiforma), relaciones, 688  
   formas de entrada de datos que proporcionan contexto para que representan tipos de entidad asociativa, 144-146  
 M-forma, tipo de entidad asociativa, 146, 182, 679. *Vea también* Tipos de entidad asociativa  
 Microsoft Access  
   combinación de agrupamiento e intersecciones, 95  
   condiciones de enlace en las columnas de fecha, 88  
   consultas anidadas en las cláusulas FROM, 313-314, 318, 319, 320  
   consultas 1-M que se pueden actualizar, 350-351  
   CREATE VIEW, sentencia en el modo de consulta de SQL, 92, 341  
   DELETE, sentencia que usa el estilo del operador de escritorio, 14

diagramas de bases de datos relacionales, 140  
 enlace inexacta con el operador LIKE, 87  
 EXCEPT, palabra clave no apoyada, 306  
 expresiones en las cláusulas SELECT y WHERE, 86  
 forma proporcionada por, 10  
 herramienta gráfica para acceder, 8-9  
 información proporcionada por, 10  
 intersección externo completo no apoyado, 300  
 intersecciones externas que preceden a los internos, 302  
 múltiples sentencias SELECT en lugar de consultas anidadas, 332  
 posibilidad de actualización de una vista de tablas múltiples, 349  
 que permite la definición de las reglas de autorización, 488  
 SELECT, diferencias de sintaxis, 131  
 SELECT, ejemplos de la sentencia, 82  
 VBA integrada con, 10  
 ventana de definición de relaciones, 8  
 ventana de definición de tablas, en la, 7-8  
 ventana de diseño de consultas, 9, 60-61, 93  
 ventana de relaciones, 53-54  
 Microsoft Office, 14  
 Microsoft Office Visio Professional 2003. *Vea* Visio 2003 Professional  
 Microsoft, participación de mercado del software de una base de datos empresarial, 13  
 Microsoft SQL Server, lenguaje Transact-SQL, 10  
 middleware, 609, 610-611, 687  
   que se añade para reducir la contención, 607  
   tipos disponibles comercialmente, 611  
 middleware de bases de datos de objetos, 651-652, 655, 688  
 miembros de dimensiones, 562  
 minería de datos, 558-559, 681  
 minimalismo de LHS, 223  
 MIN, función, 92  
 MINUS, palabra clave, 113, 131, 306, 308  
 MOD, función, 257  
 modelado de datos  
   construcción, 182  
   notaciones, 37  
   plantillas en Visio 2003 Professional, 37  
 modelado de datos conceptuales  
   fase de desarrollo de la base de datos, 29-30  
   Student Loan Limited, para, 455-460  
 modelado orientado a objetos, 157  
 modelador de Datos AllInfusion ERWin, 36  
 modeladores de datos, 33  
 modelo de cascada, 25, 695  
 modelo de consultas ANSI 92 en Microsoft Access, 87  
 modelo de datos, 26, 681. *Vea también* Modelo de entorno de interacción; modelo del proceso finalización, 179  
 refinamiento, 173  
 modelo de datos relacionales, 690  
 modelo de interacción del entorno, 26, 683. *Vea también* Modelo de datos; Modelo de procesos  
 modelo de madurez para almacenes de datos, 560  
 modelo de procesos, 26, 690. *Vea también* Modelo de datos; Modelo de interacción del ambiente  
 Modelo de Relación de Entidades, 29, 147-150  
 modelo empresarial de datos (EDM, por sus siglas en inglés), 485, 556, 683  
 modelo jerárquico de datos, 12  
 modelo relacional  
   jerarquías relationales que no tienen apoyo directo, 191  
   relaciones M-N y, 54  
 modelos, 559  
 modelos empresariales, 497-498  
 modificación de la vista, 344-345, 694-695  
 MODIFY, palabra clave, 76  
 modo de consulta de Microsoft Access, 341  
 MOLAP (OLAP multidimensional), 589, 687  
 monitoreo de actividad, 28  
 monitores de procesamiento de la transacción, 611, 693-694  
 Month, función, 127  
 MonthName, función, 127  
 movimiento mecánico de un disco, 251  
 MS SQL Server 2000, sintaxis de diferencias SELECT, 131  
 MTTR Advisor, 533  
 MULTISET, tipo, 657, 658, 661  
 Multisets, 657  
 MVD (dependencia multivaluada), 235, 687-688.  
   *Vea también* FD (dependencias funcionales); Independencia de relación  
 MySQL, 14, 502-503

## N

naturaleza iterativa del modelado de datos, 173  
 navegación del DBMS, 12  
 necesidades de información del negocio, análisis de las, 168-170  
 NEW (nuevo) palabra clave, 411  
 nivel de aislamiento comprometido en lectura, 536, 537  
 nivel de aislamiento de lectura que se puede repetir, 536, 537  
 nivel de aislamiento no comprometido en lectura, 536  
 nivel de aislamiento que se puede hacer en serie, 536, 537  
 nivel de salida en informes jerárquicos, 361-362  
 nivel externo  
   Arquitectura de Tres Esquemas, de la, 340  
   descripción de la base de datos, de la, 15, 16  
   niveles de clasificación, asignación de objetos, 487  
   niveles en una sentencia CREATE DIMENSION, 571, 572  
 NO ACTION (sin acción) palabra clave, 55  
 nodo de raíz, llave de, 688  
 nodo hijo  
   clave de, 688  
   en una forma de, 431  
 nodo padre en una forma, 431  
 nodos, 684  
   capacidad de árbol BTREE, 260  
   contenido del árbol BTREE, 260-261  
   dividiendo, 261-262  
   en una forma de diagrama jerárquico, 431  
 nodos de servidor multiprocesador, 619  
 nodos terminales  
   B+tree, en un, 264  
   estructura Btree, en una, 260  
 nombre del paquete, 401-402  
 nombres de alias  
   en la cláusula ORDER BY, 97  
   requeridos para una auto-unión, 109  
 nombres de funciones, apoyados por UML, 157  
 nombres globales de bases de datos, 630  
 nombres, múltiples, 594  
 no primarias. *Vea* Columnas no llave  
 normalización, 35, 223, 688  
   análisis de objetivos de, 238  
   cuestiones acerca de, 237-238  
   en la fase de diseño lógico de la base de datos, 30  
   en las tablas de Student Loan Limited, 462-464  
   rol en el proceso de desarrollo de la base de datos, 237-238  
 normalización de datos. *Vea* Normalización  
 notación de sintaxis, 75  
 notación del diagrama de clase del UML, 157-159  
 NOT EXISTS operador, 308, 309, 310-312  
 NOT FINAL palabra clave, 657  
 %NotFound atributo cursor, 399  
 NOT IN operador, 305-306, 308  
 NOT INSTANTIABLE, palabra clave, 657  
 NOT NULL palabra clave, 52  
 NOT operador lógico, 57  
 NOT tabla de verdad, 322  
 núcleo relacional en DBMS relationales a objetos, 653  
 NUMBER tipo de datos, 381  
 NUMERIC tipo de dato, 47  
 Null lado derecho, escribiendo un FD con, 238

número único de secuencia en bitácora (LSN, por sus siglas en inglés), 527

## O

OBJECT, palabra reservada, 665  
objeto grande de caracteres. *Vea CLOB*  
objetos, 644, 688  
acceso por medio de interfases, 645  
asignación de niveles de clasificación, 487  
colecciones de, 644  
identificadores únicos para, 644  
persistentes, 654  
usados en un paquete, 401-402  
objetos complejos, manipulación de los, 662-664  
objetos de aplicación, por medio de restricciones de autorización, 488  
objetos de bases de datos, dependencias entre los, 493-494  
objetos de secuencia en Oracle, 77, 169  
objetos persistentes, 654  
objetos referenciados, eliminación de, 494  
ODBC. *Vea Conectividad de base de datos abierta*  
ODMG. (Grupo de Administración de Bases de Datos de Objetos), 654, 655  
ODMG orientado a objetos, 655  
OLAP (procesamiento analítico en línea), 688  
extensión en SQL:2003, 583  
tecnologías de almacenamiento, 589-591  
OLD, palabra clave, 411  
ON DELETE CASCADE, acción, 416  
ON DELETE, cláusula, 55  
ON ERROR, sentencia, 526  
ONLY, palabra clave, 664, 669  
ON UPDATE, cláusula, 55  
operación de deshacer en la bitácora de transacciones, 527  
operación delete en una vista que se puede actualizar, 348  
operaciones complejas, programación de las bases de datos  
lenguajes necesarios para las, 377-378  
operaciones de bases de datos, 10-11  
operaciones de bitácora, generadas en el momento de reiniciar, 531-532  
operaciones de diferencias, 309-312  
operaciones de intersección  
adicionales, consultas con, 273  
combinación con una operación diferente, 309-312  
divididas, 617  
innecesarias, eliminación de las, 273  
operaciones de intersección externa, 334-335  
operaciones de intersección partida, 634  
operaciones de serie, 270  
operador AND, 57, 89  
operador BETWEEN\_AND, 88, 679  
operador de apertura de candado, 523  
operador de bloqueos, 523  
operador de cierre transitivo, 377  
operador de cubo de datos de pivote, 566, 689.  
*Vea también* Cubo de datos  
operador de cubo de datos enrollados, 566, 578, 691. *Vea también* Operador de cubo de datos *drill-down*  
operador de cubo de datos slice, 564-565, 566, 692. *Vea también* Operador de datos de cubo dice  
operador de cubos que despliegan mayor detalle, 565-566, 683. *Vea también* Operador de cubos que generaliza  
operador de diferencias, 63-64, 65, 68, 305, 682  
operador de dividir, 66-67, 69, 314-315, 682  
operador de intersección, 59-61, 68, 686  
operador de intersección externa, 61-63, 68, 689  
operador de intersección externa completa, 62, 684  
operador de intersección externa de un solo lado, 62-63, 306  
operador de intersección natural, 59-60, 688. *Vea*  
Operador de intersección-equí; Operador de intersección  
operador de referenciado, 664

operador de producto, 68  
operador de proyectos, 56-57, 68, 690  
operador de puntos  
expresiones de camino en Oracle 10g, en, 669  
sentencia SELECT, en la, 664  
operador de una intersección de un solo lado, 298-299, 688  
consulta de subforma con, 358  
en Oracle 8i SQL, 335-336  
mezcla con otras intersecciones, 301-302, 336  
unión con dos, 300-301  
operador de unión, 63, 64-65, 68, 694  
operador extendido de producto vectorial, 57-59, 683  
operador, intersect, 68  
operadores de comparación, 85, 88  
empalme de índices, en el, 265  
PL/SQL, en, 382, 383  
operadores de serie, 63-65  
operadores de serie tradicionales, 63-65, 111-113, 693  
optimización  
decisiones, 268, 271-274  
procesamiento de consultas distribuidas, 631-632  
software, 252  
tecnología, 12  
optimización de consulta, 16, 252, 268-274  
optimización de consulta de intersección de inicio por máquinas ROLAP, 590  
optimización dinámica versus estática, 631  
optimizador de consultas en Oracle RAC, 618  
OQL (Lenguaje de Consultas de Objetos, por sus siglas en inglés), 654  
OR, condición con un valor nulo, 322  
OR, operador, 57  
comparado con el operador IN, 93  
mezcla con AND en una expresión lógica, 89  
OR, palabra clave en un especificación de evento disparado, 404  
OR, tabla de la verdad, 322  
Oracle  
combinación de agrupamiento e intersecciones, 95  
condiciones de enlace de las columnas de fecha, 88  
CREATE DIMENSION, sentencia, 571-574  
CREATE INDEX, sentencia, 379-280  
CREATE VIEW, sentencia, 340-341  
disparadores, 403-414  
disparadores encimados, 415  
enlace inexacto con el operador LIKE, 87  
expresiones en las cláusulas SELECT y WHERE, 87  
extensión propietaria que usa el símbolo (+), 337  
extensiones propietarias para disparadores, 403  
generación de valor automático por medio de objetos de secuencia, 77  
lenguaje PL/SQL, 10  
lenguaje propietario de programación de bases de datos para, 380  
MINUS, palabra clave, 306  
paquetes predefinidos, 399  
participación de mercado del software empresarial de bases de datos, 380  
procedimiento de ejecución del disparador, 414-416  
RAC (Grupos de Aplicaciones Reales, por sus siglas en inglés), 618-619, 689  
SELECT, diferencias de sintaxis, 131  
SELECT, ejemplos de sentencias proporcionados para, 82  
sensibilidad del caso de, 88  
tablas del catálogo de, 495, 496  
término de una sentencia, 84  
transparencias de bases de datos distribuidas, 628-630  
vistas de intersecciones actualizables, 349, 372-373  
Oracle 10g  
características de bases de datos de objetos en, 664-672

CREATE TABLE, sintaxis, 76  
extensión de las sentencias de seguridad de SQL:2003, 488  
limitaciones en las restricciones CHECK, 491  
proceso de recuperación, 532-533  
que apoya a los tipos definidos por el usuario, 665-668  
tablas capturadas en, 668-670  
vistas materializadas en, 583-585  
Oracle Designer 10g, 36  
Oracle Enterprise Manager, 61  
Oracle 8i  
intersección externa completa que no tiene apoyo directo, 300  
notación de intersección externa, 335-337  
orden descendente, 96  
ORDER BY, cláusula, 95-96, 100  
orientación de procesos de bases de datos operacionales, 555  
orientación por temas en los almacenes de datos, 555  
Organización Internacional de Estándares (ISO, por sus siglas en inglés), 496, 686  
organizaciones, efectos en la tecnología de bases de datos, 17-20  
ORGANIZATION INDEX, cláusula, 279  
Origination Loan Form, 451, 453  
ERD para, 455, 456  
requerimientos de datos para, 467-468  
OTHERS, excepción, 390, 391  
OVERRIDING, palabra clave  
Oracle 10g, en, 665  
Tipo ColorPoint, en, 657

P

padre. *Vea* Súptipo  
páginas. *Vea también* Registros físicos como hotspots dependientes de un sistema, 534  
páginas de paridad, 284  
pagos, repartidos por Student Loan Limited, 453-454  
paquetes, 689  
PL/SQL, en, 398-402  
predefinidos en Oracle, 399  
SQL:2003 en, 655  
paradoja, 14  
paralelismo partido con la opción DPF, 619  
parámetro de entrada (IN, por su nombre en inglés), 389  
parámetro de entrada-salida (IN OUT, por su nombre en inglés), 389  
parámetro de salida (OUT), 389  
parámetros de PL/SQL, procedimientos, 389  
paréntesis, agrupamiento de las condiciones explícito, 89  
parte fija de un formato jerárquico, 353, 354  
partición  
entre procesadores dentro de la arquitectura SN, 616  
por las máquinas ROLAP, 590  
partición determinada por el DBA, apoyada por la opción DPF, 619  
participación de mercado del software empresarial de bases de datos, 13  
pata de cuervo, notación,  
comparada con otras notaciones, 156  
ERD, para, 136  
resumen de, 150  
símbolos que representan cardinalidades, 138  
patrón de estrella, 271  
patrones, detectados en la minería de datos, 558  
perfiles de aplicación, 252, 253, 255-256, 679  
componentes de los, 255  
definición de Student Loan Limited, 464-465  
perfiles de tablas, 252, 253-255, 693  
componentes de, 254  
deficiencias de estimación, 272  
definición de Student Loan Limited, 464-465  
periodo de tiempo de carga, 594, 595, 687. *Vea*  
*también* Período de tiempo válido

periodo de tiempo válido, 594, 595, 694. *Vea también* Período de tiempo de carga; Tiempo de transacción  
 permisos de usuarios y grupos, ventana, 488, 489  
 persistencia de los datos, 4  
 personalización, apoyada por herramientas de desarrollo de una aplicación de bases de datos, 377  
 pesos a la par, 500-501  
 PL/SQL, 380  
   bloque, 387  
   cursores en, 395-398  
   ejecución de sentencias en bloques anónimos, 386-388  
   ejemplos de sentencias, 380-386  
   funciones, 392-394  
   fundamentos, 380-386  
   operadores, 380  
   paquetes, 398-402  
   procedimientos, 389-391  
 plan de transición, 26  
 planeación de datos, desempeñada por DA, 497-498  
 planeación de los sistemas de negocios. *Vea también* Sistemas de información  
 planes de acceso, 269, 679  
   desplegados gráficos de, 271  
   determinación, 379  
   distribución, 631  
   ejecución, 271  
   evaluación, 269-271  
   recopilación, 494  
   variaciones de, 270  
 planes de acceso distribuidos, 631. *Vea también* Planes de acceso  
 plantilla de relación de entidades en Visio Professional, 37, 38  
 pistas, que afectan la selección de planes de acceso, 271-272  
 polimorfismo, 647-649, 689  
 política de vencimiento, control de los atolladeros, 524  
 portabilidad, 378  
 POSITIVE, tipo de datos, 381  
 PostgreSQL, 14  
 PowerDesigner, 10, 36  
 PR. *Vea* Registros físicos  
 preguntas críticas, 80, 101-103  
 preocupaciones de división del procesamiento, 612  
 prestadores de préstamos GSL, 450  
 préstamos a alumnos, sistema de información para su procesamiento, 24  
 préstamos GSL sin subsidio, 450  
 préstamos GSL subsidiados, 450  
 primario. *Vea* Columna de llaves  
 Primer Formato Normal (1FN, por sus siglas en inglés), 223, 224  
 principio de simplicidad, durante la búsqueda de tipos de entidad, 169  
 principios orientados a objetos, 649  
 privilegio ANALYZE ANY, 489  
 privilegios  
   asignación para control de acceso discrecional, 486  
   especificación en la sentencia GRANT, 487  
   especificación para vistas, 486  
 privilegios ALTER, 489  
 privilegios de objetos en Oracle 10g, 488, 489  
 privilegios del sistema, 488, 489  
 problema de lectura fantasma, 522, 689  
 problemas de desempeño de *middleware* de objetos, 651  
 problemas de diferencias, 305  
   fórmulas limitadas de SQL para, 305-308  
   uso de consultas anidadas de tipo II para, 308-312  
 problemas de división  
   avanzados, 317-320  
   DISTINCT dentro de COUNT, con, 318  
   formulación, 314-320  
   intersección, con un, 317-318  
   sencillos, 315-317  
 problemas de especificación, resueltos por medio de la documentación, 179

problemas narrativos, análisis, 168  
 procedimiento de detección lineal, 258, 259  
 procedimiento de impresión en Oracle 10g, 665  
 procedimiento de la base de datos, 388  
 procedimiento de lenguajes de programación, 388  
 procedimiento de preparación  
   en SQL:2003 CLI, 379  
 procedimiento de síntesis simple, 229-232, 692  
 procedimientos, 388  
   administración de DBMS de, 388  
   contra de funciones en PL/SQL, 392  
   SQL:2003, en, 657  
 procedimientos almacenados, 344-402, 493-495, 692  
 procedimientos de ejecución de disparador simple, 414-415  
 procedimientos de ejecución del disparador, 414-416, 694  
 procedimientos de refresco, 583  
 procesador de consultas de objetos, 653  
 procesamiento  
   ambientes para DBMS, 503  
   ubicación de, 31  
   volumen como un impedimento para suavizar la conversión, 467  
 procesamiento centralizado de consultas, 631  
 procesamiento cliente-servidor  
   objetos se localicen en distintas computadoras, permite que, 648  
   razones empresariales, para, 606-607, 609  
 procesamiento de bases de datos paralelas, 615-619  
   comparado con el procesamiento de bases de datos distribuidas, 620  
   costos elevados de, 608  
   razones empresariales para el, 607-608, 609  
   redes de comunicación empleadas en, 631  
 procesamiento de compromiso distribuido, 633-635  
 procesamiento de consultas distribuidas, 631-632  
 procesamiento de consultas, interpretación cuando las tablas contienen valores nulos, 320  
 procesamiento de transacción, 7, 10, 11, 503, 693  
   en contra del apoyo a las decisiones, 554  
   distribuido, 632-635  
 procesamiento de transacciones distribuidas, 632-635. *Vea también* Procesamiento de transacciones  
 procesamiento distribuido, 16, 607, 682. *Vea también* Arquitectura cliente-servidor  
 procesamiento paralelo, 283-284  
 procesamiento por lotes, 377  
 proceso de división en tablas más pequeñas, 225  
 proceso de escritura de bitácora (LGWR, por sus siglas en inglés), 618  
 proceso de escritura de las bases de datos (DBWR, por sus siglas en inglés) en Oracle RAC, 618  
 proceso de evaluación conceptual, 79, 681  
   cláusulas SELECT, para las, 97-101  
   intersección desigual aplicado a una problema de diferencia, 307-308  
 proceso de jerarquía analítica, 500-501, 679  
 procesos del evento, 377  
 procesos de refresco, 594-596  
 producto cartesiano, 57  
 producto vectorial, estilo de, 103, 298, 681  
   combinación con el estilo del operador de intersección, 108  
   comparación con los estilos del operador de intersección, 109  
 producto vectorial, operador de, 98  
 programadores, 18  
 propagación de la actualización  
   AFTER ROW, disparador para, 408-409  
   disparadores para, 403  
 propiedad aislada, 518, 679  
 propiedad atómica, 518, 679  
 propiedad compartida de las bases de datos, 4  
 propiedad consistente, 518, 679  
 propiedad de cercanía de los índices en cluster, 274-275  
 propiedad de conversión para una serie de tiempo, 564  
 propiedad de fecha de inicio de una serie de tiempo, 564  
 propiedad de periodicidad de una serie de tiempo, 564  
 propiedad de tipos de datos de una serie de tiempo, 564  
 propiedad durable, 519, 679  
 propiedad persistente para las bases de datos, 4  
 propiedad transitiva, 226, 686. *Vea también* FD (dependencias funcionales, por sus siglas en inglés); Dependencias transitivas  
 propiedades, almacenadas en las herramientas CASE, 35  
 propiedades de calendario de una serie de tiempo, 564  
 protocolo, 252  
 protocolo de acceso de objetos simples (SOAP, por sus siglas en inglés), 614, 615  
 protocolo de bloqueos de dos sases. *Vea* 2PL (candidato de dos fases, por sus siglas en inglés)  
 protocolo de compromiso de dos fases (2PC, por sus siglas en inglés), 634-635, 694  
 protocolo de registro de escritura hacia adelante, 529, 695. *Vea también* Metodología de actualización diferida; Metodología de actualización inmediata  
 protocolo primario de copia, 633, 689  
 prototipos, 26, 32, 690  
 proveedores de servicio  
   en la arquitectura de Web Services, 614  
   para préstamos GSL, 450  
 proyecto del sistema R, 80  
 puntero en un nodo Btree, 260  
 punto de revisión de consistencia de caché, 528, 680  
 punto y coma (:)  
   en PL/SQL, 380  
   en SQL\*Plus, 386  
 puntos de referencia, 502, 679  
 puntos de revisión borrosos, 528, 684. *Vea* Puntos de revisión  
 puntos de verificación, 527-528, 528, 680. *Vea también* Puntos borrosos de verificación; Puntos incrementales de verificación  
 puntos incrementales de revisión, 528, 533, 685. *Vea también* Puntos de revisión

## Q

Quinto Formato Normal (5NF, por sus siglas en inglés), 223, 236-237

## R

RAC (Grupos Reales de Aplicación, por sus siglas en inglés), 618-619, 689  
 RAID (deslizamientos redundantes de discos independientes, por sus siglas en inglés), 263, 690  
 RAID, almacenamiento, 283-284  
 RAID, controlador, 283  
 RAID-1, 284, 690  
 RAID-5, 284, 690  
 raíz de una familia de subtablas, 662  
 REAL, tipo de datos, 47  
 realización de prototipos  
   herramientas, 36  
   Visio, en, 39  
 recuperación, 519  
   administración, 526-533  
   herramientas, 527-528  
   procesos, 529-533  
   transparencia, 519, 560  
 recursos de cómputo, minimización de los, 251  
 red, distribución de software e información, 16-17  
 red partida, 634  
 redes computacionales. *Vea* Lenguaje de programación de redes computacionales. *Vea* Lenguajes de programación  
 reducción de tiempo, costo de la, 608  
 redundancias, 219

- análisis de tablas para, 220-223  
arquitecturas RAID, en, 284  
formas de, 527  
MVD que guía a, 235  
redundancias excesivas, eliminación de las, 220  
reescritura de consulta, 583, 585, 690. *Vea también*  
    Vistas materializadas  
    que involucra bloques SELECT, 587-588  
    que enlaza los requerimientos para, 585-587  
    que usa vistas materializadas, 590  
referencias de la vista, proceso de consultas con, 344-346  
REFERENCIAS, privilegio, 487  
REF, función, 669  
refinamientos a ERD, 173-178  
Refresh Models Wizard en Visio, 38-39  
registro de compromiso, 634  
registro de servicio en una arquitectura de servicios Web, 614  
registros. *Vea también* Registros duplicados  
    agrupación de todos los, 94  
    continuos, 92  
    eliminación de uno a más, 114-115  
    eliminación del duplicado, 96-97  
     fusión como una manera rápida de interseccesar tablas, 276  
     inserción de uno a la vez, 113  
     inserción en consultas 1-M que se pueden actualizar, 351-352  
     manipulación de las familias de subtablas, 662  
     que contienen valores excluidos de resultados, 321  
     tabla, en una, 46  
registros bloqueados, 523  
registros duplicados. *Vea también* Eliminación de registros, 104  
    eliminación, 96-97  
     representación de, 571-574  
registros físicos, 260, 689  
    direcciones, 257  
    lectura paralela de, 283  
    transferencia, 250-251  
registros referenciados, 690  
    acciones en, 415, 416, 679  
    actualización de llaves primarias de, 54  
    eliminar y actualizar acciones para, 54-55  
    reglas acerca de, 691  
regla de conexión del tipo de relación/entidad, 152  
regla de conexión del tipo de relación/relación, 152  
regla de entidades débiles, 152  
    resolución, 154  
    violación de, 153  
regla de jerarquía de generalización, 461  
regla de llave extranjera redundante, 152, 153  
    ER Assistant, y, 155  
    resolución, 154  
    violación de, 153  
regla de llave primaria, 152  
regla de nombre en una entidad, 152  
regla de nombres de atributo heredados, 152  
regla de participación en una unidad, 152, 153  
regla de tipos de entidad, 184, 461  
regla de relación de identificación, 152, 685. *Vea también* Entidades débiles  
    resolución, 154  
    violación de la, 153  
regla del nombre del atributo, 152  
reglas básicas para la conversión de ERD, 183-187  
reglas de completado, 152, 153  
reglas de conversión  
    conversión de ERD en tablas relacionales, para la, 183-187  
    conversión de ERD Student Loan Limited, 461  
    ejemplo completo, 193-195  
reglas de diagrama, 152-155  
reglas de evento-condición-acción. *Vea* Eventos del disparador  
reglas de integridad  
    aplicación de las, 50-53  
    definición de las, 49-50  
reglas del negocio  
    definición, 27  
    representadas en una ERD, 151-152  
reglas para fijación de nombre, 152  
reinicio  
    metodologías de actualización, 530  
    operaciones de la bitácora generadas en el, 531-532  
relación, 690  
relación cifrada, 137  
relación de identificación, 141, 150, 156  
relación de uno a muchos (1-M). *Vea* 1-M (uno a muchos), relación  
relación de valores sencillos, 139  
relación funcional, 139  
relación opcional, 139, 689. *Vea también* Relación obligatoria  
relación ternaria, 145, 693  
relación unaria. *Vea* Relaciones autoreferenciadas relacionales, 664  
    DBMS relacionales a objetos para, 652-654  
    en SQL:2003, 491, 656-658  
    métodos, 657  
    preconstruidos, 652  
relaciones, 151, 152, 690. *Vea también* Relaciones 1-M (uno a muchos); Relaciones M-N  
    (muchos a muchos); Integridad referencial  
    añadir, 169-170, 172-173  
    aspectos de, 141-147  
    base de datos de un hospital, en una, 6  
    base de datos de una universidad, en una, 5  
    base de datos del servicio de aguas, en una, 5  
    conexión de los tipos de entidad con, 433-434  
    despliegue con una jerarquía explotada, 563  
    entre entidades, 4  
    entre tablas, 47  
    ERD, en, 23, 136-137  
    identificación de las ocultas, 559  
    indicación de conexiones entre las tablas, 6  
    mal ubicadas y faltantes, 181  
    patrones para, 142-146  
    redundantes, 182  
    relaciones auto referenciadas, 52-53, 143-144, 691  
    aplicación de reglas para conversión, 186  
    consultas que involucran, 377  
    que se representan en la ventana de Relaciones, 54  
relaciones de composición, con apoyo del UML, 159  
relaciones de precedencia entre las formas, 441  
relaciones faltantes, 181  
relaciones muchos a muchos (M-N). *Vea* Relaciones M-N (muchos a muchos)  
relaciones M-N (muchos a muchos), 54, 139, 156, 687. *Vea también* Relaciones 1-M (uno a muchos), Relaciones  
    atributos, con, 142-143, 150  
    que representan UML, 158  
    que reutilizan el sistema de estrella para, 568-569  
    que se transforman en una entidad de tipo asociativo, 176-177, 178  
    que usa una relación 1-M en lugar de, 182  
    reemplazo por una entidad de tipo asociativo y dos relaciones 1-M que se identifican, 146  
    regla de conversión, 164, 185, 186  
    regla que se aplica al ERD de Student Loan Limited, 461  
relaciones redundantes, 182  
relaciones reflexivas. *Vea* Relaciones auto-referenciadas  
relaciones ubicadas erróneamente, 181  
relaciones 1-M opcionales, 188-189  
reorganización de archivos hash, 259  
reparto automático, 619  
repetir la operación en la bitácora de la transacción, 527  
réplica en RAID, 284  
replicación, 31  
repositorio. *Vea* Diccionarios de datos; Diccionario de recursos de información (IRD, por sus siglas en inglés)  
repositorio de Microsoft, 496  
representación de la base de datos, conversión de una sentencia de problema en, 102-103  
representación gráfica de integridad referencial, 53-54  
representación relacional de datos de ventas, 561  
representaciones de tiempo en el esquema estrella, 569-571  
requerimientos de datos, 29  
    enlace de columnas y tablas, 102  
    pasos para el Loan Origination Form, 467-468  
requerimientos de enlace para la reescritura de consultas, 585-587  
requerimientos de información para la base de datos de los servicios de agua, 171-173  
requerimientos de las bases de datos, utilización de los formatos como, 429  
resolución de sinónimos y homónimos, 441-442  
resolución del conflicto en la vista del proceso de integración, 32  
RESOURCE, función, 488  
respaldo, 528  
respaldo de bases de datos, 528  
restricción basada en los valores, 221  
restricción cláusula de tiempo de  
    SQL en, 538  
    SQL:2003, sintaxis para, 551-552  
restricción de completado, 148, 149, 150, 680-681  
restricción difícil, 413. *Vea también* Restricciones  
restricción neutral a los valores, 221  
restricción, operador, 56-57, 68, 690  
restricción: selección de los niveles de, 27  
restricción suave, 413  
restricciones. *Vea también* Restricciones para las bases de datos  
    duras, 413  
    especificación del tiempo para complejas, 537-538  
    especificación en una sentencia de CREATE DIMENSION, 573  
    múltiples tablas y cálculos estadísticos, de, 491  
proceso de actualización, 595-596  
proceso de diseño de la base de datos física, de, 252  
resultante de las teclas y dominios, 237  
restrictiones de acceso de fuente, 595, 596  
restrictiones de bases de datos, 221. *Vea también* Restricciones  
restrictiones de clave externa (FK, por sus siglas en inglés), 221  
restrictiones de comparación de atributos, 151  
restrictiones de integridad, 7, 490-493  
    disparadores para complejas, 402-403  
    tiempo de reforzamiento, 537-539  
restrictiones de llave primaria (PK, por sus siglas en inglés), 221  
restrictiones de refresco, 690  
restrictiones de transición  
    BEFORE, disparadores empleados para, 411  
    disparadores que refuerzan, 403  
restrictiones de valor null, 151, 152  
restrictiones numéricas, 381  
RESTRICT, cláusula, 488  
RESTRICT, palabra clave, 55  
resumen de relaciones en un perfil de tabla, 254  
resumen incorrecto, 522, 685  
resumir, operador, 65-66, 68, 693  
retiradas inconsistentes, 521-522  
reusabilidad, apoyada por lenguajes de programación orientados a objetos, 649  
Reverse Engineer Wizard en Visio, 39  
revisión de restricciones diferidas, 537, 682  
revisión de restricciones, duración de las transacciones y, 533  
revisión no razonable, 593  
REVOKE, sentencia, 81, 488  
RIGHT JOIN, palabra clave, 299  
ROLAP (OLAP Relacional), 589-590, 691  
rollback parcial, 539  
ROLLBACK, sentencia  
    PL/SQL, procedimiento, en el, 389, 390  
    transacción, en una, 517  
ROLLBACK TO SAVEPOINT, palabras claves, 539  
ROLLUP, operador, 578-580, 582, 691  
rompedores de objetos, 611  
ROW, palabra clave, 662

- S
- salidas del diseño de bases de datos físicas, 252  
SAN (Redes de Área de Almacenamiento, por sus siglas en inglés), 284, 692  
sangría en informes jerárquicos, 359-360  
sangría, estilo de, 9  
SAVEPOINT, sentencia, 539  
Save Point, sentencias, 552  
Save Points, 539, 691  
SCOPE, cláusula  
  Oracle 10g, en, 666  
  SQL:2003, en, 663  
sección ejecutable en un bloque PL/SQL, 387  
secuencia en Oracle, 489  
segundo formato normal (2NF, por sus siglas en inglés), 223, 225, 226-227, 691  
seguridad, 486-489  
  base de datos, 486, 682  
  nivel flexible proporcionado por las vistas, 340  
seguridad de la base de datos, 486, 682  
selección de índices, 274-280, 685  
  definición del problema, 274-276  
  dificultades de, 277  
  entradas y salidas de, 274-275  
  número exponencial de opciones, 277  
  reglas de, 277-280  
  soluciones transaccionales de, 276-277  
  tablas de la base de datos de la universidad, para las, 278-279  
SELECT, cláusula, 86-87  
Select, operador. Vea Restricción, Operador  
SELECT, privilegio, 487  
SELECT, sentencia (s), 82-84  
  aleatoria, 95-96  
  auto-intersecciones, 109-110  
  compatibilidad en la unión de tablas, 111-113  
  con expresiones de camino y operador dereferenciado, 664  
  con la función REF en Oracle 10g, 669  
  de SQL, 8  
  dentro de la sentencia INSERT, 114  
  en problemas de tabla sencilla, 84-89  
  enlace de condiciones en la columna de fechas, 88  
  enlace de múltiples tablas con el estilo del producto vectorial, 103-106  
  enlace de tablas, 89-91  
  enlace exacto, en, 87-88  
  evaluación de los registros por medio de la cláusula WHERE, 85  
  evaluación para ubicar valores nulos, 89  
  expresiones en las cláusulas SELECT y WHERE, 86-87  
  expresiones lógicas complejas, 89  
  formato simplificado de, 82  
  funciones en las expresiones en , 393-394  
  mejora en la apariencia de los resultados, 95-97  
  múltiples en Microsoft Access en lugar de consultas anidadas, 332-333  
  nuevas funciones agregadas empleadas en, 583  
  obtención del identificador de objetos, 663  
  proceso de evaluación conceptual para, 97-101  
  que muestra todas las columnas, 86  
  que produce el resultado de una operación CUBE, 577-578  
  que produce los resultados de una operación ROLLUP, 579-580  
  resumen de las tablas con GROUP BY y HAVING, 91-95  
  resumen de sintaxis de SQL:2003, 333-334  
  resumen de sintaxis simplificado de SQL:2003, 128-130  
  vistas en, 342-344  
SELECT ANY, privilegio, 489  
selector en una sentencia CASE, 384  
semi-intersección, 625  
semi-intersección, operador, 691  
sentencia ALTER TABLE, 76  
sentencia de cuenta  
  archivos hash estáticos, 259, 684  
  área de almacenamiento de la red (SANs), 284, 692  
  cursores estáticos en PL/SQL, 395  
  independencia estadística, 232  
  integración estática, 379, 648, 679  
  interfase de nivel de sentencia, 378-379, 692  
  modelado de datos de, 458-459  
  nivel de almacenamiento de las bases de datos, 250-251  
  Stencil en Visio Professional, 37  
  tecnologías de almacenamiento, 589-591  
  sentencia de problema, 25, 102-103  
  sentencia del disparador, 423  
  sentencias de asignación, 382  
  sentencias de control de bases de datos, 81  
  sentencias de índices, 279  
  sentencias de iteración, 385-386  
  sentencias de manipulación de bases de datos, 81  
  sentencias de manipulación de datos en un disparador, 415-416  
  sentencias de modificación, 113-115  
  sentencias de procedimiento, combinación con el acceso a la base de datos, 376  
  sentencias SELECT anidadas, 333-334  
SEQUEL, lenguaje, 80  
serie de índices en un Btree, 264  
serie de secuencia en un B+tree, 264  
series  
  operación de diferencia entre dos, 311-312  
  tablas como, 57  
series, que enlazan a las subseries, 88  
series constantes, 27, 381  
series de tiempo, 564  
Servicio Global de Caché (GCS, por sus siglas en inglés) en Oracle RAC, 618  
servicios de enlistado de bienes raíces, 643  
servidor de la base de datos, 611. *Vea también servidores*  
servidor middleware, 612  
servidores, 16, 388, 606, 691. *Vea también Servidores de aplicación; Servidores de medios; Servidores de medios especializados*  
servidores de aplicación en una arquitectura de capas múltiples, 612, 614  
servidores de medios, 655  
servidores de medios especializados, 650-651  
servidores remotos, administración de procedimientos almacenados en los, 388  
SET, comando, 386  
SET CONSTRAINTS, sentencia, 538-539, 552  
SET DEFAULT, palabras clave, 55  
SET NULL, palabra clave, 55  
SET TRANSACTION, sentencia  
  colocada antes de una sentencia START TRANSACTION, 536, 537  
  SQL:2003, sintaxis para, 552  
SGA (Área Global Compartida, por sus siglas en inglés), 618  
SHOW, comando, 386  
signo de interrogación ?, que enlaza a cualquier carácter sencillo, 88  
símbolo de porcentaje % en un carácter comodín, 87  
símbolos y palabras reservados en PL/SQL, 380  
simplificación de consultas modificadas, 346  
simplificación de la intersección, 269  
simplificación de tareas como beneficio de vistas, 340  
simula, 649  
sinónimos, 441, 442, 693.  
  *Vea también Homónimos*  
sistema de entrada de pedidos, transacciones en, 516  
sistema de nómina de una universidad, transacciones en un, 516-517  
sistema del diccionario de recursos de la información (IRDS, por sus siglas en inglés), 496  
sistema físico, 24  
sistema(s), 24, 693  
sistemas de administración de bases de datos. Vea DBMS  
sistemas de fuente, cambio de datos de los, 591  
sistemas de fuentes externas para datos de almacenamiento, 591  
sistemas de información, 24, 25-26, 686  
  planeación, 497-498, 686  
  profesionales, 17-18  
sistemas de legado, a partir de datos sucios, 593  
sistemas de procesamiento de archivos, 12  
sistemas internos de fuentes para los datos del almacén de datos, 591  
sitio de coordinación para un procesamiento de compromiso de dos fases, 634  
sitios para DBMS distribuidos, 620  
sitios participantes para un procesamiento de compromiso de dos fases, 634  
SMALLINT, tipo de datos, 47, 381  
Smalltalk, 649  
SOAP, 614, 615  
sobre flujo, 687  
software  
  cliente, 17  
  mantenimiento de, 14, 612  
  código abierto, 502  
  DBMS de código abierto, 502  
  optimización, 252  
  servidor, 17  
  tercer, 11  
software de bases de datos, mercado actual para, 13-14  
software de terceros para DBMS, 11  
software del cliente, 17  
software del servidor, 17  
solicitante de servicio en una arquitectura de servicios Web, 614  
solicitudes globales, 620, 630  
soluciones iAnywhere, 14  
SPOOL, comando, 386  
SQL (Lenguaje de Consultas Estructurado, por sus siglas en inglés), 7, 692  
  adopción como estándar, 12  
  alcance de, 81-82  
CREATE TABLE, sentencia, 46  
dominios, 490-491  
escritura del código portátil, 81  
estándares para, 80-81  
formulación de consultas con, 79-116  
historia de, 80-81  
operadores de serie tradicional en, 111-113  
realizar la división en, 315  
SELECT, sentencia, 8  
sentencias de modificación, 113-115  
sentencias seleccionadas, 81  
tipos de datos, 490  
SQL:2003, 74, 81, 692  
  apoyo directo a la organización de jerarquías, 659  
aserciones, 491-493  
características de bases de datos de objetos en, 655-664  
clasificación de disparadores, 403  
CONNECT, sentencia especificada por, 379  
CREATE ASSERTION, sintaxis, 512  
CREATE DOMAIN, sentencias, 490  
CREATE DOMAIN, sintaxis, 512  
CREATE ROLE, sintaxis, 511-512  
CREATE TABLE, sintaxis de sentencia, 74-76  
CREATE VIEW, sintaxis de sentencia, 372  
DELETE, resumen de sintaxis de sentencia, 131  
Disparadores encimados, 415-416  
DROP ASSERTION, sintaxis, 512  
DROP DOWN, sintaxis, 512  
DROP ROLE, para, 511-512  
DROP VIEW, sintaxis de sentencia, 372  
estándares objeto-relacionales en, 654  
estilos de lenguajes para integrar un lenguaje de procedimiento con SQL, 378-379  
funciones y procedimientos, 657  
generalización de jerarquías para tablas que se apoyan directamente por, 191  
INSERT, resumen del sintaxis de la sentencia, 130  
integración estática y dinámica especificada por, 379

- limitaciones y métodos, 657  
 niveles de aislamiento, 537  
 OLAP, extensión, 583  
 operadores de diferencias, 306  
 partes y paquetes, 656  
 procedimiento de ejecución del disparador especificado por, 414  
 que da apoyo a los estilos de definiciones de tablas, 659  
 SAVEPOINT, sentencias, 539, 552  
 SELECT anidada, sintaxis de sentencia, 333-334  
 SELECT simplificado, sintaxis de sentencia, 128-130  
 sentencias de índices que no tienen apoyo, 279  
 sentencias de seguridad, 487-488  
 sentencias que involucran cursor, 380  
**SERIALIZABLE**, como nivel de aislamiento predeterminado, 537  
 SET CONSTRAINT, sintaxis de sentencia, 552  
 SET TRANSACTION, sintaxis, 552  
 sintaxis de cláusula de restricción de tiempo, 551-552  
 sintaxis de operaciones de intersección externa, 334-335  
 sintaxis de sentencia del disparador, 423  
 SQL/PSM definido por, 378  
 tablas de catálogos en, 495-496  
 tablas de la verdad, 321-322  
 tipos definidos por el usuario, 491  
 tipos distintos, 490  
 UPDATE, resumen de sintaxis de la sentencia, 130  
**USING**, cláusula proporcionada por, 380  
 XML, apoyo, 671  
 SQL central, 81, 692  
 SQL incrustado, 683. *Vea también* SQL Standalone  
 SQL incrustado, sentencias, 378  
 SQL-86, estándar, 81  
 SQL-89, modo de consulta en Microsoft Access 2002 y 2003, 341  
 SQL-92, estándar, 74, 81, 659  
 SQL\*Plus, 386  
 SQL> prompt, 386  
 SQL propio, 692  
 SQL/PSM, 378  
 SQL ROLLBACK, sentencia, 526  
 SQL SELECT, sentencia. *Vea* SELECT, sentencias  
 SQL, sentencias  
     asociación con planes de acceso, 379  
     evaluación de los disparadores, 406  
     procesamiento de los resultados de las, 379-380  
 START TRANSACTION, sentencia, 516-517  
 Stripe, 283, 693  
 Striping, 283-284, 590, 690, 693  
 Student Loan Limited, 450  
     campos de formato e informe, 471-474  
     CREATE TABLE, sentencias, 474-477  
     decisiones de implementación, 467  
     desarrollo de las aplicaciones, 467-469  
     diseño de bases de datos físicas, 465-467  
     flujo de trabajo para un sistema propuesto para, 450-455  
     modelado de datos conceptuales, 455-460  
     refinamiento del esquema conceptual para, 461-464  
 subclases, 646  
 subconsulta. *Vea* Consultas anidadas  
 subconsultas correlacionadas, 308  
 subfórmula, 353, 354, 357-358, 359, 693  
 subseries de vistas, 686  
 subseries horizontales, 622, 684  
 subseries horizontales derivadas, 684  
 subseries verticales, 684  
 subtablas que no tienen apoyo en Oracle 10g, 666, 668  
 subtipos, 3-20, 148, 693  
     UM, función, 92, 323  
 superclave mínima, 50  
 superllave, 49, 693  
     mínima, 50, 680  
 supertipo, 148, 178, 693  
 sustituibilidad de captura, 670-671  
 SYSTEM GENERATED, palabras clave, 660
- T
- tabla base, intersección con una vista, 343-344  
 tabla bloqueada, 523  
 tabla de conservación de claves, 372, 373, 686  
 tabla de dimensión de tiempo, 569  
 tabla máxima, 662  
 tabla padre  
     insertar un registro en, 352  
     Microsoft Access, en, 350  
**TABLA** tipo de colección, 670  
 tabla verdadera AND, 322  
 tabla virtual o derivada, 340  
 tablas, 6, 46-47, 693. *Vea también* Tablas anidadas  
     álgebra relacional que aplica a, 56  
     combinación, 59, 102, 300  
     conexiones entre, 47-49  
     intersecciones múltiples, 103-109  
     intersecciones, 89-91  
     involucradas en relaciones de subtablas, 661  
     relación con formas jerárquicas, 354-355  
     uso de una muestra para analizar problemas difíciles, 101  
 tablas actualizables, 356  
 tablas anidadas, 224. *Vea* tablas  
     en Oracle 10g, 671  
     soportadas por SQL: 2003, 661  
 tablas capturadas  
     en Oracle 10g, 668-670  
     proporcionadas por SQL: 2003, 660  
     obtención de todas las columnas de las, 670  
     apoyadas por Oracle 10g, 665-666  
 tablas de bases de datos. *Vea* Tablas  
 tablas de catálogo, 495-496  
 tablas de dimensiones, 682  
     cambios de tarifas para, 594  
     diseño para el retiro, 568  
     en un esquema estrella, 567  
     representación de tiempo para, 570  
 tablas de hechos, 683. *Vea también* Esquemas de copo de nieve; Esquema de estrella  
     bitácora de estado de eventos múltiples, 570  
     cambio de tasas para, 594  
     esquemas estrellas, en, 567  
     normalización, 568  
     participación en operaciones internacionales, 569  
 tablas de la base de datos de una universidad  
     contenedores de, 82-84, 85  
     selección de índices para, 278-279  
 tablas de verdad, 321-322  
 tablas mutantes, 416-417  
 tablas relacionales, conversión de ERD en, 183-195  
 tareas, afectadas por la división  
     del procesamiento, 610  
 tasa de error del almacén de datos, 595  
 teclado ALL, 113  
 tecnología comercial para bases de datos en paralelo, 617-619  
 tecnología de bases de datos  
     efectos en la vida cotidiana, 3  
     evolución de los, 12-13  
     impactos organizacionales de, 17-20  
 tecnología de bases de datos de segunda generación, 691  
 tecnología de bases de datos de tercera generación, 693  
 tecnología de cuarta generación para bases de datos, 684  
 tecnología de Fusión de Caché, 618  
 tecnología de Información Sólida, 14  
 tecnología de primera generación para bases de datos, 683  
 tecnologías, mejora, 540-542  
 tercer formato normal (3NF, por sus siglas en inglés), 225-229, 693  
 terminología alternativa para bases de datos relacionales, 49  
 terminología multidimensional, 562-564  
 terminología orientada a series, 49  
 terminología orientada a tablas para bases de datos relacionales, 49
- terminología orientada al registro, 49  
**TEXT**, tipo de datos, 73  
 texto cifrado, 487, 683  
 texto plano, 487, 683  
 tiempo, clasificación de los disparadores para, 403  
 tiempo de carga, 687  
 tiempo de espera, 687  
 tiempo de interacción entre usuarios, 535-536  
 tiempo de respuesta  
     medición, 31  
     minimizar, 251  
     transacción relacionada con, 520  
 tiempo de transacción, 694  
 tiempo representativo para la recuperación (MTTR, por sus siglas en inglés), 533  
 tiempo válido, 694  
 tiempos de actualización, distintos, 594  
**TIME**, tipo de datos, 47  
**Timeout\_On\_Resource**, excepción, 391  
 tipo de datos AutoNumber, 77, 169  
 tipo de datos BOOLEAN, 47, 381  
 tipo de datos *timestamp*, 569  
 tipo de entidad de forma, 432, 684  
 tipo de punto, 656  
 tipos abstractos, 657  
 tipos de datos, 46-47, 681  
     combinación con Oracle, 382  
     estándar para DBMS, 642  
     lenguajes de programación, en, 379  
     necesidad de nuevos, 505  
     PL/SQL, 381  
     SQL, en, 490  
 tipos de datos anclados, 493  
 tipos de datos primitivos, 382  
 tipos de entidad, 683  
     ERD, en, 29, 136, 137  
     expansión, 173-174, 175, 178  
     formato de factura, para el, 435  
     identificación, 169, 170, 171-172, 431  
     involucrados en relaciones múltiples, 170  
     reglas para conexión, 432  
     símbolo de la notación de Pata de Cuervo para, 150  
     transformación de atributos en, 173, 174, 178  
     transformación en jerarquías de generalización, 177-178  
     Visio, en, 37, 38  
 tipos definidos por el usuario  
     implementados por los vendedores  
     principales de DBMS  
     que se definen en Oracle 10g, 665-668  
 tipos de Informix definidos por el usuario, 652  
 tipos de Oracle definidos por el usuario, 652  
 tipos de referencia, 666  
 tipos de registro, 659-660, 665-666  
 tipos distintos, 490, 491  
 tipos preconstruidos definidos por el usuario, 652  
 To\_char, función, 87  
 To\_Date, función, 381  
 To\_number, función, 87  
 Too\_Many\_Rows, excepción, 391  
 totales de resumen, que representan vía cube, 561-562  
 TPC (consejo de procesamiento de transacciones, por sus siglas en inglés), 693  
 TPC-C prueba de evaluación (introducción de pedidos), 520  
 traducción de consultas, fases de, 268-271  
 transacción global, 620  
 transacciones, 10, 516, 693. *Vea también* Transacciones anidadas  
     aborted, 526  
     administración personalizada, 542  
     anidadas, 542  
     como definidas por el usuario, 516  
     cuestiones de diseño, 533-539  
     ejemplo, 516-518  
     en un sistema de entrada de pedidos, 516  
     enrolamiento parcial de las, 539  
     global, 620  
     húerfanas, 594  
     número de concurrentes apoyadas, 519  
     propiedades de las, 518-519

recorte de la duración de, 517-518  
 SQL, definición de las sentencias, 516-518  
 tipos de, 516  
 transacciones anidadas, 542. Ver transacciones  
 transacciones autónomas, 542  
 transacciones de bases de datos. Vea Transacciones  
 transacciones definidas por el usuario, 516  
 transacciones huérfanas, 594  
 transformación de consultas, durante la  
     optimización de consultas, 268-269  
 transitividad, ley de la, 686  
 transparencia, 624  
     en las bases de datos distribuidas de Oracle, 628-630  
     niveles de, 624-626  
     para DBMS, 519  
     para el procesamiento de las bases de datos  
         distribuidas, 624-630  
 transparencia de fragmentación, 626, 628, 684. Vea  
     también transparencia de mapeo local;  
     Transparencia de ubicación  
 transparencia de la concurrencia, 519, 681  
 transparencia de mapeo local, 628, 629, 687. Vea  
     también Transparencia de fragmentación;  
     Fragmentos  
 transparencia de ubicación, 627-628, 687.  
 Vea también Transparencia de  
     fragmentación; Fragmentos  
 transportación en la fase de preparación del  
     mantenimiento del almacén de datos, 592, 593  
 TRIGGER, privilegio, 487  
 %TYPE, palabra clave, 382  
 TYPE, sentencia, 381

## U

UDDI, 614, 615  
 UML (Lenguaje de Modelado Unificado, por sus  
     siglas en inglés), notación de diagrama de  
     clase de, 157-159  
 UNDER, palabra clave  
     en el tipo ColorPoint, 657  
     en Oracle 10g, 671  
 unidad de procesamiento central. Vea Coordinación  
     centralizada del uso del CPU  
     control de concurrencia, de, 633  
     protocolo del compromiso de dos fase, de, 635  
 UNION, consulta, 112  
 UNIQUE, palabra clave, 50-51  
 Unix, 13  
 UPDATE, disparadores  
     disparadores encimados para, 415  
     lineamientos para, 495  
 UPDATE, privilegio, 487  
 UPDATE, sentencia, 114  
     activación de los disparadores para cada, 405  
     costo del mantenimiento de los índices como  
         resultado de, 277  
     resumen de sintaxis de SQL:2003, 130  
     uso de la sentencia SELECT para obtener el  
         identificador de objetos, 663  
 UPDATE ANY, privilegio, 489  
 USER GENERATED, palabras clave, 660  
 USING, cláusula, 380  
 uso del parámetro en los procedimientos y  
     funciones, 493  
 usuarios de poder, 18, 689. Vea también Usuarios  
     indirectos; Usuarios paramétricos  
 usuarios funcionales, 17, 18

usuarios indirectos, 18, 685. Vea también Usuarios  
     paramétricos; usuarios de poder  
 usuarios paramétricos, 18, 689. Vea también  
     Usuarios indirectos; Usuarios de poder

## V

valor de claves en un nodo Btree, 260  
 valor fantasma, 521  
 valor null en PL/SQL, 381  
 valores constantes, 381  
 valores de enlace, combinación de múltiples tablas  
     que usan, 48-49  
 valores de importancia, cálculo de, 501  
 valores de parámetro en, distribución de para la  
     selección de índice, 277  
 valores, faltantes, 594  
 valores null, 50, 688  
     efectos de, 320-324  
     en llaves foráneas, 52  
     especificaciones para columnas, 113  
     llevar a complicaciones en la evaluación de los  
         resultados de consultas, 188  
     para registros que no enlacen, 61  
     pruebas para usar SELECT, 89  
 valores preestablecidos, 55, 151, 152  
 valores únicos de columna, 92  
 valores únicos, que se generan de las llaves  
     primarias, 76-77  
 VALUE, función, 670  
 VALUES, cláusula, 113  
 VARCHAR, tipo de datos, 47  
 VARCHAR2, tipo de datos, 73, 381  
 variable de correlación con la función REF, 669  
 variable (repetida) parte de un formato jerárquico,  
     353, 354  
 variables, 644  
 variables de instancias, 644  
 variables de la clase, 644  
 variables dependientes, 232  
 variables independientes, 232  
 variaciones de regla en una notación ERD, 156  
 VARRAY, constructor, 668  
 ventana de relación en Microsoft Access, 53-54  
 ver disparadores, restricciones de tabla que no tiene  
     mutaciones, 417  
 verificación de completado, 593  
 verificación de la captura, 648  
 verificación de tipos fuertes, 648, 693  
 versiones, documentadas por las herramientas  
     CASE, 35  
 Visible Analyst, 7, 6, 36  
 Visio 2003 Professional, 37-39  
 Visio Professional, ERD creada por, 11  
 vista de agrupación, uso de la consulta, 343  
 vista de la intersección, 686  
 vista de tablas múltiples  
     definición, 341  
     que usa las consultas, 342-343  
 vista de tablas sencillas, 341  
 vista externa, 683. Vea también Esquemas;  
     Arquitectura de Tres Esquemas  
 vistas, 16, 340, 694. Vea también Esquemas  
     externos; Vistas materializadas  
     definición, 340-342  
     desempeño y, 340  
     en formas jerárquicas, 353-359  
     en informes, 359-362  
     en las sentencias SELECT, 342-344  
     integración en un paso, 689  
     modificaciones a, 347

que protegen a los usuarios de detalles del  
     fragmento, 628  
 seguridad proporcionada por, 340  
     uso de la actualización, 346-353  
     uso para la recuperación, 342-346  
 vistas actualizables, 346-353, 694  
 vistas de interacción actualizables  
     en oracle, 349, 372-373  
     mapeo de los registros de, 686  
 vistas de tablas múltiples que se pueden actualizar,  
     349-353  
 vistas jerárquicas, 671  
 vistas materializadas, 583-585  
     conciencia del usuario acerca de las, 584-585  
     Oracle, en, 489  
     reescritura de consulta que usa, 590  
     sustitución por tablas de hechos y  
         dimensiones, 585

vistas que se pueden actualizar de tablas sencillas,  
     346-349  
 vistas sólo de lectura, 346, 347, 690  
 Visual Basic para Aplicaciones (VBA, por sus siglas  
     en inglés), 9-10, 39  
 Visual Studio .Net Enterprise Architect, 36, 37  
 vocabulario  
     desarrollo de un común, 27  
     estandarización, 442  
 vocabulario común, 27  
 volatilidad de los dispositivos de almacenamiento  
     de datos, 526

## W

WHEN, cláusula, 403  
 WHERE, cláusula  
     comparada con HAVING, 93  
     evaluación de los registros por medio de, 85  
     expresiones en, 86-87  
     referenciada en una columna indizada, 265  
     UPDATE e INSERT, sentencias que violan las  
         cláusulas de una vista, 349  
     verificación posterior de los resultados  
         intermedios, 101  
 WHILE LOOP, sentencia, 385, 386  
 WITH CHECK OPTION, 349, 372, 695  
 WITH GRANT OPTION, palabra clave, 488  
 WITH RECURSIVE, cláusula, 377  
 WITHADMIN, cláusula, 487  
 WSDL (Lenguaje de Descripción de Servicios Web,  
     por sus siglas en inglés), 614, 615  
 WSFL (Lenguaje de Flujo de Servicios Web, por  
     sus siglas en inglés), 614, 615  
 WWW (World Wide Web, por sus siglas en inglés),  
     695

## X

XML (Lenguaje de marcación extensible, por sus  
     siglas en inglés), 614, 671, 695  
     apoyo a documentos, 671-672  
     esquemas, 671, 672  
     operadores, 672  
 XMLType, tipo de datos, 671-672

## Y

Year, función  
     en Microsoft Access, 86, 468