FUNDAMENTOS DE BASES DE DATOS

Cuarta edición

FUNDAMENTOS DE BASES DE DATOS

Cuarta edición

Abraham Silberschatz
Bell Laboratories

Henry F. Korth
Bell Laboratories

S. Sudarshan
Instituto Indio de Tecnología, Bombay

Traducción
FERNANDO SÁENZ PÉREZ
ANTONIO GARCÍA CORDERO
CAROLINA LÓPEZ MARTÍNEZ
LUIS MIGUEL SÁNCHEZ BREA
OLGA MATA GÓMEZ
M.ª VICTORIA GONZÁLEZ DEL CAMPO RODRÍGUEZ BARBERO
Universidad Complutense de Madrid

Revisión técnica
LUIS GRAU FERNÁNDEZ
Universidad Nacional de Educación a Distancia

MADRID • BUENOS AIRES • CARACAS • GUATEMALA • LISBOA • MÉXICO
NUEVA YORK • PANAMÁ • SAN JUAN • SANTAFÉ DE BOGOTÁ • SANTIAGO • SÃO PAULO
AUCKLAND • HAMBURGO • LONDRES • MILÁN • MONTREAL • NUEVA DELHI • PARÍS
SAN FRANCISCO • SIDNEY • SINGAPUR • ST. LOUIS • TOKIO • TORONTO

FUNDAMENTOS DE BASES DE DATOS. Cuarta edición

Traducido de la cuarta edición en inglés de
Database System Concepts

ISBN: 0-07-228363-7

ISBN: 84-481-3654-3
Depósito legal: M.

Editora: Concepción Fernández Madrid
Editora de mesa: Susana Santos Prieto
Cubierta: DIMA
Compuesto en FER
Impreso en:

IMPRESO EN ESPAÑA - PRINTED IN SPAIN

En memoria de mi padre, Joseph Silberschatz
y de mis abuelos Stepha y Aaron Resenblum.

Avi Silberschatz

A mi esposa, Joan,
mis hijos, Abigail y Joseph,
y mis padres, Henry y Frances

Hank Korth

A mi esposa, Sita,
mi hijo, Madhur,
y mi madre, Indira.

S. Sudarshan

CONTENIDO BREVE

CONTENIDO

PREFACIO

LA gestión de bases de datos ha evolucionado desde una aplicación informática especializada hasta una parte esencial de un entorno informático moderno y, como resultado, el conocimiento acerca de los sistemas de bases de datos se ha convertido en una parte esencial en la enseñanza de la informática. En este libro se presentan los conceptos fundamentales de la administración de bases de datos. Estos conceptos incluyen aspectos de diseño de bases de datos, lenguajes de bases de datos e implementación de sistemas de bases de datos.

Este libro está orientado a un primer curso de bases de datos para niveles técnicos y superiores. Además del material básico para un primer curso, el texto también contiene temas que pueden usarse como complemento del curso o como material introductorio de un curso avanzado.

En este libro se asume que se dispone de los conocimientos elementales sobre estructuras de datos básicas, organización de computadoras y un lenguaje de programación de alto nivel (tipo Pascal). Los conceptos se presentan usando descripciones intuitivas, muchas de las cuales están basadas en el ejemplo propuesto de una empresa bancaria. Se tratan los resultados teóricos importantes, pero se omiten las demostraciones formales. Las notas bibliográficas contienen referencias a artículos de investigación en los que los resultados se presentaron y probaron, y también referencias a material para otras lecturas. En lugar de demostraciones, se usan figuras y ejemplos para sugerir por qué se espera que los resultados en cuestión sean ciertos.

Los conceptos fundamentales y algoritmos tratados en este libro se basan habitualmente en los que se usan en la actualidad en sistemas de bases de datos existentes, comerciales o experimentales. Nuestro deseo es presentar estos conceptos y algoritmos como un conjunto general que no esté ligado a un sistema de bases de datos particular. En la Parte 8 se discuten detalles de sistemas de bases de datos comerciales.

En esta cuarta edición de *Fundamentos de bases de datos* se ha mantenido el estilo global de las primeras tres ediciones, a la vez que se ha tenido en cuenta la evolución de la gestión de bases de datos. Se han añadido varios capítulos nuevos para tratar nuevas tecnologías. Cada capítulo se ha corregido y la mayoría se ha modificado ampliamente. Se describirán los cambios con detalle en breve.

ORGANIZACIÓN

El texto está organizado en ocho partes principales más dos apéndices:

- **Visión general** (Capítulo 1). En el Capítulo 1 se proporciona una visión general de la naturaleza y propósito de los sistemas de bases de datos. Se explica cómo se ha desarrollado el concepto de sistema de bases de datos, cuáles son las características usuales de los sistemas de bases de datos, lo que proporciona al usuario un sistema de bases de datos y cómo un sistema de bases de datos se comunica con los sistemas operativos. También se introduce una aplicación de bases de datos de ejemplo: una empresa bancaria que consta de muchas sucursales. Este ejemplo se usa a lo largo de todo el libro. Este capítulo es histórico, explicativo y motivador por naturaleza.
- **Modelos de datos** (Capítulos 2 y 3). En el Capítulo 2 se presenta el modelo entidad-relación. Este modelo proporciona una visión de alto nivel de los resultados de un diseño de base de datos y de los problemas que se encuentran en la captura de la semántica de las aplicaciones realistas que contienen las restricciones de un modelo de datos. El Capítulo 3 se centra en el modelo de datos relacional, tratando la relevancia del álgebra relacional y el cálculo relacional.
- **Bases de datos relacionales** (Capítulos 4 al 7). El Capítulo 4 se centra en el lenguaje relacional orientado al usuario de mayor influencia: SQL. El Capítulo 5 cubre otros dos lenguajes relacionales, QBE y Datalog. En estos dos capítulos se describe la manipulación de datos: consultas, actualizaciones, inserciones y borrados. Los algoritmos y las cuestiones de diseño

se relegan a capítulos posteriores. Así, estos capítulos son adecuados para aquellas personas o para las clases de nivel más bajo en donde se desee aprender qué es un sistema de bases de datos, sin entrar en detalles sobre los algoritmos internos y estructuras que contienen.

En el Capítulo 6 se presentan las restricciones desde el punto de vista de la integridad de las bases de datos. En el Capítulo 7 se muestra cómo se pueden usar las restricciones en el diseño de una base de datos relacional. En el Capítulo 6 se presentan la integridad referencial; mecanismos para el mantenimiento de la integridad, tales como disparadores y asertos, y mecanismos de autorización. El tema de este capítulo es la protección de las bases de datos contra daños accidentales y daños intencionados.

En el Capítulo 7 se introduce la teoría del diseño de bases de datos relacionales. Se trata la teoría de las dependencias funcionales y la normalización, con énfasis en la motivación y el significado intuitivo de cada forma normal. También se describe en detalle el proceso de diseño de bases de datos.

- **Bases de datos basadas en objetos y XML** (Capítulos 8 al 10). El Capítulo 8 trata las bases de datos orientadas a objetos. En él se introducen los conceptos de la programación orientada a objetos y se muestra cómo estos conceptos constituyen la base para un modelo de datos. No se asume un conocimiento previo de lenguajes orientados a objetos. El Capítulo 9 trata las bases de datos relacionales de objetos, y muestra cómo la norma SQL:1999 extiende el modelo de datos relacional para incluir características de la programación orientada a objetos, tales como la herencia, los tipos complejos y la identidad de objeto.

  En el Capítulo 10 se trata la norma XML para representación de datos, el cual está experimentando un uso cada vez mayor en la comunicación de datos y en el almacenamiento de tipos de datos complejos. El capítulo también describe lenguajes de consulta para XML.

- **Almacenamiento de datos y consultas** (Capítulos 11 al 14). En el Capítulo 11 se estudian los discos, archivos y estructuras de un sistema de archivos y la correspondencia de datos relacionales y de objetos con un sistema de archivos. En el Capítulo 12 se presentan varias técnicas de acceso a los datos, incluyendo la asociación, los índices de árboles B+ y los índices de archivos en retícula. Los capítulos 13 y 14 tratan los algoritmos de evaluación de consultas y optimización de consultas basados en transformación de consultas preservando la equivalencia.

  Estos capítulos están orientados a personas que desean conocer los componentes de almacenamiento y consulta internos de una base de datos.

- **Gestión de transacciones** (Capítulos 15 al 17). El Capítulo 15 se centra en los fundamentos de un sistema de procesamiento de transacciones, incluyendo la atomicidad de las transacciones, la consistencia, el aislamiento y la durabilidad, y también la noción de secuencialidad.

  El Capítulo 16 se centra en el control de concurrencia y se presentan varias técnicas que aseguran la secuencialidad, incluyendo los bloqueos, las marcas temporales y técnicas optimistas (de validación). Los temas de interbloqueo se tratan también en este capítulo. El Capítulo 17 aborda las técnicas principales para asegurar la ejecución correcta de transacciones a pesar de las caídas del sistema y los fallos de disco. Estas técnicas incluyen el registro histórico, paginación en la sombra, puntos de revisión y volcados de la base de datos.

- **Arquitectura de un sistema de bases de datos** (Capítulos 18 al 20). El Capítulo 18 trata la arquitectura de un sistema informático y en él se describe la influencia de los sistemas informáticos subyacentes en los sistemas de bases de datos. Se discuten los sistemas centralizados, los sistemas cliente-servidor, las arquitecturas paralelas y distribuidas, y los tipos de redes. En el Capítulo 19 se estudian los sistemas de bases de datos distribuidas, revisando los aspectos de diseño de bases de datos, gestión de las transacciones y evaluación y optimización de consultas en el contexto de los sistemas de bases de datos distribuidas. El capítulo también trata aspectos de la disponibilidad del sistema durante fallos y describe el sistema de directorios LDAP.

  En el capítulo 20, acerca de las bases de datos paralelas, se exploran varias técnicas de paralelización, incluyendo paralelismo de E/S, paralelismo entre consultas y en consultas, y paralelismo entre operaciones y en operaciones. También se describe el diseño de sistemas paralelos.

- **Otros temas** (Capítulos 21 al 24). El Capítulo 21 trata el desarrollo y administración de aplicaciones de bases de datos. Los temas incluyen las interfaces de las bases de datos, en particular las interfaces Web, el ajuste de rendimiento, los programas de prueba, la estandarización y los aspectos de las bases de datos en el comercio electrónico. El Capítulo 22 presenta

técnicas de consulta, incluyendo sistemas de ayuda a la toma de decisiones y recuperación de la información. Los temas tratados en el área de la ayuda a la toma de decisiones incluyen las técnicas de procesamiento analítico interactivo (OLAP, Online Analytical Processing), el soporte de SQL:1999 para OLAP, recopilación de datos y almacenes de datos. El capítulo también describe técnicas de recuperación de información para la consulta de datos textuales, incluyendo técnicas basadas en hipervínculos usadas en los motores de búsqueda Web.

El Capítulo 23 trata tipos de datos avanzados y nuevas aplicaciones, incluyendo datos temporales, datos espaciales y geográficos, datos multimedia, y aspectos de la gestión de las bases de datos móviles y personales. Finalmente, el Capítulo 24 trata el procesamiento avanzado de transacciones. Se estudian los monitores de procesamiento de transacciones, los sistemas de transacciones de alto rendimiento, los sistemas de transacciones de tiempo real, y los flujos de datos transaccionales.

- **Estudios de casos** (Capítulos 25 al 27). En esta parte presentamos estudios de casos de tres sistemas de bases de datos comerciales: Oracle, IBM DB2 y Microsoft SQL Server. Estos capítulos esbozan características únicas de cada uno de los productos y describen su estructura interna. Proporcionan una gran cantidad de información interesante sobre los productos respectivos, y ayudan a ver cómo las diferentes técnicas de implementación descritas en las partes anteriores se usan en sistemas reales. También se tratan aspectos prácticos en el diseño de sistemas reales.
- **Apéndices en línea.** Aunque la mayoría de las aplicaciones de bases de datos modernas usen, bien el modelo relacional o bien el modelo orientado a objetos, los modelos de datos de redes y jerárquico están en uso todavía. En beneficio de los lectores que deseen aprender estos modelos de datos se proporcionan apéndices que describen los modelos de redes y jerárquico, en los Apéndices A y B, respectivamente. Los apéndices sólo están disponibles en Internet (http://www.bell-labs.com/topic/books/db-book).

  El Apéndice C describe el diseño avanzado de bases de datos relacionales, incluyendo la teoría de dependencias multivaloradas ¿Multivaluadas?, las dependencias de reunión y las formas normales de proyección-reunión y dominio-clave. Este apéndice es útil para quienes deseen el tratamiento del diseño de bases de datos relacionales en más detalle, y para profesores que deseen explicarlo en sus asignaturas. Este apéndice está también sólo disponible en Internet, en la página Web del libro.

LA CUARTA EDICIÓN

La producción de esta cuarta edición se ha guiado por muchos comentarios y sugerencias referidos a las ediciones anteriores, junto con las propias observaciones en la enseñanza en el IIT de Bombay, y por el análisis de las direcciones que la tecnología de bases de datos está tomando.

El procedimiento básico fue reescribir el material en cada capítulo, actualizando el material más antiguo, añadiendo discusiones en desarrollos recientes en la tecnología de bases de datos, y mejorando las descripciones de los temas que los estudiantes encontraron difíciles de comprender. Cada capítulo tiene ahora una lista de términos de repaso, que pueden ayudar a asimilar los temas clave tratados en cada capítulo. Se ha añadido también una nueva sección al final de la mayoría de los capítulos que proporciona información sobre herramientas software referidas al tema del capítulo. También se han añadido nuevos ejercicios y se han actualizado las referencias.

Se ha incluido un nuevo capítulo que trata XML, y tres capítulos de estudio de los sistemas de bases de datos comerciales líderes: Oracle, IBM DB2 y Microsoft SQL Server.

Los capítulos se han organizado en varias partes y se han reorganizado los contenidos de varios de ellos. En beneficio de aquellos lectores familiarizados con la tercera edición se explican a continuación los principales cambios.

- **Modelo entidad-relación.** Se ha mejorado el tratamiento del modelo entidad-relación (E-R). Se han añadido nuevos ejemplos y algunos se han cambiado para dar una mejor intuición al lector. Se ha incluido un resumen de notaciones E-R alternativas, junto con un nuevo apartado sobre UML.
- **Bases de datos relacionales.** El tratamiento de SQL en el Capítulo 4 ahora se refiere al estándar SQL:1999, que se aprobó después de la publicación de la tercera edición de este libro. El

tratamiento de SQL se ha ampliado significativamente para incluir la cláusula **with**, para un tratamiento ampliado de SQL incorporado y el tratamiento de ODBC y JDBC, cuyo uso ha aumentado notablemente en los últimos años. La parte del capítulo 5 dedicada a Quel se ha eliminado, ya que no se usa ampliamente debido al poco uso que actualmente se hace de este lenguaje. El tratamiento de QBE se ha revisado para eliminar algunas ambigüedades y para añadir el tratamiento de la versión de QBE usada en la base de datos Microsoft Access.

El Capítulo 6 trata ahora de las restricciones de integridad y de la seguridad. El tratamiento de la seguridad, ubicado en la edición anterior en el Capítulo 19, se ha trasladado al Capítulo 6. El Capítulo 6 también trata los disparadores. El Capítulo 7 aborda el diseño de las bases de datos relacionales y las formas normales. La discusión de las dependencias funcionales, ubicada en la edición anterior en el Capítulo 6, se ha trasladado al Capítulo 7. El Capítulo 7 se ha remodelado significativamente, proporcionando varios algoritmos para las dependencias funcionales y un tratamiento extendido del proceso general del diseño de bases de datos. Los axiomas para la inferencia de las dependencias multivaloradas, las formas normales FNRP y FNCD se han trasladado al apéndice.

- **Bases de datos basadas en objetos.** Se ha mejorado el tratamiento de la orientación a objetos del Capítulo 8, y se ha actualizado la discusión de ODMG. Se ha actualizado el tratamiento de las bases de datos relacionales orientadas a objetos del Capítulo 9 y, en particular, el estándar SQL:1999, reemplaza a SQL extendido usado en la tercera edición.
- **XML.** El Capítulo 10, que trata XML, es un nuevo capítulo de la cuarta edición.
- **Almacenamiento, indexación y procesamiento de consultas.** Se ha actualizado el tratamiento del almacenamiento y de las estructuras de archivos del Capítulo 11; este fue el Capítulo 10 en la tercera edición. Muchas características de las unidades de disco y de otros mecanismos de almacenamiento han cambiado en gran medida con el paso de los años, y su tratamiento se ha actualizado correspondientemente. El tratamiento de RAID se ha actualizado para reflejar las tendencias tecnológicas. El tratamiento de diccionarios de datos (catálogos) se ha extendido.

  El Capítulo 12, sobre indexación, incluye ahora el estudio de los índices de mapa de bits; este capítulo fue el Capítulo 11 en la tercera edición. El algoritmo de inserción en árboles B^+ se ha simplificado y se ha proporcionado un pseudocódigo para su examen. La asociación dividida se ha eliminado, ya que no tiene un uso significativo.

  El tratamiento del procesamiento de consultas se ha reorganizado, con el capítulo anterior (Capítulo 12 en la tercera edición) dividido en dos capítulos, uno sobre procesamiento de consultas (Capítulo 13) y otro sobre optimización de consultas (Capítulo 14). Todos los detalles referidos a la estimación de costes y a la optimización de consultas se han trasladado al Capítulo 14, permitiendo al Capítulo 13 centrarse en los algoritmos de procesamiento de consultas. Se han eliminado varias fórmulas detalladas (y tediosas) para el cálculo del número exacto de operaciones de E/S para diferentes operaciones. El Capítulo 14 se presenta ahora con un pseudocódigo para la optimización de algoritmos, y nuevos apartados sobre la optimización de subconsultas anidadas y sobre vistas materializadas.
- **Procesamiento de transacciones.** El Capítulo 15, que proporciona una introducción a las transacciones, se ha actualizado; este capítulo era el Capítulo 13 en la tercera edición. Se han eliminado los tests de secuenciabilidad.

  El Capítulo 16, sobre el control de concurrencia, incluye un nuevo apartado sobre la implementación de los gestores de bloqueo, y otro sobre los niveles débiles de consistencia, que estaban en el Capítulo 20 de la tercera edición. Se ha ampliado el control de concurrencia de estructuras de índices, proporcionando detalles del protocolo cangrejo, que es una alternativa más simple al protocolo de enlace B, y el bloqueo de siguiente clave para evitar el problema fantasma. El Capítulo 17, que trata sobre recuperación, incluye ahora un estudio del algoritmo de recuperación ARIES. Este capítulo trata ahora los sistemas de copia de seguridad remota para proporcionar una alta disponibilidad a pesar de los fallos, característica cada vez más importante en las aplicaciones «24×7».

  Como en la tercera edición, esta organización permite a los profesores elegir entre conceptos de procesamiento de transacciones introductorios únicamente (cubiertos sólo en el Capítulo 15) u ofrecer un conocimiento detallado (basado en los Capítulos 15 al 17).
- **Arquitecturas de sistemas de bases de datos.** El Capítulo 18, que proporciona una visión general de las arquitecturas de sistemas de bases de datos, se ha actualizado para tratar la tecnología actual; esto se encontraba en el Capítulo 16 de la tercera edición. El orden del capí-

tulo de bases de datos paralelas y de los capítulos de bases de datos distribuidas se ha intercambiado. Mientras que el tratamiento de las técnicas de procesamiento de consultas de bases de datos del Capítulo 20 (que fue el Capítulo 16 en la tercera edición) es de primordial interés para quienes deseen aprender los interiores de las bases de datos, las bases de datos distribuidas, ahora tratadas en el Capítulo 19, son un tema más fundamental con el que debería estar familiarizado cualquiera que trabaje con bases de datos.

El Capítulo 19 sobre bases de datos distribuidas se ha rehecho significativamente para reducir el énfasis en la denominación y la transparencia, y para aumentar el tratamiento de la operación durante fallos, incluyendo las técnicas de control de concurrencia para proporcionar alta disponibilidad. El tratamiento del protocolo de compromiso de tres fases se ha abreviado, al tener detección distribuida de interbloqueos globales, ya que no se usa mucho en la práctica. El estudio de los aspectos de procesamiento de consultas se ha trasladado del Capítulo 20 de la tercera edición. Hay un nuevo apartado sobre los sistemas de directorio, en particular LDAP, ya que se usan ampliamente como un mecanismo para hacer disponible la información en una configuración distribuida.

- **Otros temas.** Aunque se ha modificado y actualizado el texto completo, nuestra presentación del material que tiene relación con el continuo desarrollo de bases de datos y las nuevas aplicaciones de bases de datos se tratan en cuatro nuevos capítulos, del Capítulo 21 al 24.

  El Capítulo 21 es nuevo en la cuarta edición y trata el desarrollo y administración de aplicaciones. La descripción de la construcción de interfaces Web para bases de datos, incluyendo servlets y otros mecanismos para las secuencias de comandos para el lado del servidor, es nueva. La sección sobre ajuste de rendimiento, que estaba anteriormente en el Capítulo 19, tiene nuevo material sobre la famosa regla de 5 minutos y sobre la regla de 1 minuto, así como algunos nuevos ejemplos. El tratamiento de la selección de vistas materializadas también es nuevo. El tratamiento de los programas de prueba y de los estándares se ha actualizado. Hay una nueva sección sobre comercio electrónico, centrándose en los aspectos de las bases de datos en el comercio electrónico y un nuevo apartado que trata los sistemas heredados.

  El Capítulo 22, que trata consultas avanzadas y recuperación de la información, incluye nuevo material sobre OLAP, particularmente sobre las extensiones de SQL:1999 para análisis de datos. El estudio de los almacenes de datos y de la recopilación de datos también se ha ampliado en gran medida. El tratamiento de la recuperación de la información se ha aumentado significativamente, en particular en el área de la búsqueda Web. Las versiones anteriores de este material estaban en el Capítulo 21 de la tercera edición.

  El Capítulo 23, que trata tipos de datos avanzados y nuevas aplicaciones, contiene material sobre datos temporales, datos espaciales, datos multimedia y bases de datos móviles. Este material es una versión actualizada del material que se encontraba en el Capítulo 21 de la tercera edición. El Capítulo 24, que trata el procesamiento de transacciones avanzado, contiene versiones actualizadas de los monitores TP, sistemas de flujo de datos, bases de datos en memoria principal y de tiempo real, transacciones de larga duración y gestión de transacciones en múltiples bases de datos, que aparecieron en el Capítulo 20 de la tercera edición.

NOTA PARA EL PROFESOR

El libro contiene tanto material básico como material avanzado, que podría no ser abordado en un único semestre. Se han marcado varios apartados como avanzados, usando el símbolo «**». Estos apartados se pueden omitir, si se desea, sin pérdida de continuidad.

Es posible diseñar cursos usando varios subconjuntos de los capítulos. A continuación se muestran varias posibilidades:

- El Capítulo 5 se puede omitir si los estudiantes no van a usar QBE o Datalog como parte del curso.
- Si la programación orientada a objetos se va a tratar en un curso avanzado por separado, los Capítulos 8 y 9 y el Apartado 11.9 se pueden omitir. Alternativamente, con ellos se puede constituir la base de un curso avanzado de bases de datos orientadas a objetos.
- El Capítulo 10 (XML) y el Capítulo 14 (optimización de consultas) se pueden omitir para un curso introductorio.

- Tanto el tratamiento del procesamiento de transacciones (Capítulos 15 al 17) como de la arquitectura de sistemas de bases de datos (Capítulos 18 al 20) poseen un capítulo de visión de conjunto (Capítulos 15 y 18 respectivamente), seguidos de capítulos más detallados. Se podría elegir usar los Capítulos 15 y 18, omitiendo los Capítulos 16, 17, 19 y 20, si se relegan estos capítulos para un curso avanzado.
- Los Capítulos 21 al 24 son adecuados para un curso avanzado o para autoaprendizaje de los estudiantes, aunque el apartado 21.1 se puede tratar en un primer curso de bases de datos.

Se puede encontrar un modelo de plan de estudios del curso, a partir del texto, en la página inicial Web del libro (véase el siguiente apartado).

PÁGINA WEB Y SUPLEMENTOS PARA LA ENSEÑANZA

Está disponible una página World Wide Web para este libro en el URL:

http://www.bell-labs.com/topic/books/db-book

La página Web contiene:

- Transparencias de todos los capítulos del libro.
- Respuestas a ejercicios seleccionados.
- Los tres apéndices.
- Una lista de erratas actualizada.
- Material suplementario proporcionado por usuarios del libro.

Se puede proporcionar un manual completo de soluciones sólo a las facultades. Para obtener más información sobre cómo obtener una copia del manual de soluciones envíe por favor un correo electrónico a customer.service@mcgraw-hill.com. En los Estados Unidos se puede llamar al 800-338-3987. La página Web de McGraw-Hill para este libro es:

http://www.mcgraw-hill.es/olc/silberschatz

CÓMO CONTACTAR CON LOS AUTORES Y OTROS USUARIOS

Se ha creado una lista de correo electrónico con la que los usuarios de este libro pueden comunicarse entre sí y con los autores. Si desea incorporarse a la lista, por favor mande un mensaje a db-book@research.bell-labs.com, incluyendo su nombre, afiliación, puesto y dirección de correo electrónico.

Nos hemos esforzado para eliminar erratas y problemas del texto, pero, como en los nuevos desarrollos de software, probablemente queden fallos; hay una lista de erratas actualizada accesible desde la página inicial del libro*. Agradeceríamos que se nos notificara cualquier error u omisión del libro que no se encuentre en la lista actual de erratas.

También desearíamos recibir sugerencias sobre la mejora del libro. Damos la bienvenida a aquellas contribuciones a la página Web del libro que pudiesen ser usadas por otros lectores, como ejercicios de programación, sugerencias sobre proyectos, laboratorios y tutoriales en línea, y consejos de enseñanza.

El correo electrónico se deberá dirigir a db-book@research.bell-labs.com. Cualquier otra correspondencia se debe enviar a Avi Silberschatz, Bell Laboratories, Room 2T-310, 600 Mountain Avenue, Murray Hill, NJ 07974, EE.UU.

* N. del T. Todas las erratas de la versión original en inglés incluidas en esta página Web en el momento de finalizar la traducción se han corregido en este libro.

AGRADECIMIENTOS

Esta edición se ha beneficiado de los muchos y útiles comentarios que nos han proporcionado los muchos estudiantes que han usado la tercera edición. Además, gran cantidad de personas nos han escrito o hablado acerca del libro, y nos han ofrecido sugerencias y comentarios. Aunque no podemos mencionar aquí a todas, agradecemos especialmente a las siguientes:

- Phil Bernhard, Instituto de Tecnología de Florida; Eitan M. Gurari, Universidad del estado de Ohio; Irwin Levinstein, Universidad Old Dominion; Ling Liu, Instituto de Tecnología de Georgia; Ami Motro, Universidad George Mason; Bhagirath Narahari, Meral Ozsoyoglu, Universidad Case Western Reserve; y Odinaldo Rodríguez, King's College de Londres; que sirvieron como revisores del libro y cuyos comentarios nos ayudaron en gran medida en la formulación de esta cuarta edición.
- Soumen Chakrabarti, Sharad Mehrotra, Krithi Ramamritham, Mike Reiter, Sunita Sarawagi, N. L. Sarda y Dilys Thomas, por su amplia y valiosa realimentación sobre varios capítulos del libro.
- Phil Bohannon, por escribir el primer borrador del Capítulo 10 describiendo XML.
- Hakan Jakobsson (Oracle), Sriram Padmanabbhan (IBM) y César Galindo-Legareia, Goetz Graefe, José A. Blakeley, Kalen Delaney, Michael Rys, Michael Zwilling, Sameet Agarwal, Thomas Casey (todos de Microsoft), por escribir los apéndices sobre los sistemas de bases de datos Oracle, IBM DB2 y Microsoft SQL Server.
- Yuri Breitbart, por su ayuda con el capítulo sobre bases de datos distribuidas; Mark Reiter, por su ayuda con los apartados de seguridad; y Jim Melton, por las aclaraciones sobre SQL:1999.
- Marilyn Turnamian y Nandprasad Joshi, cuya excelente asistencia secretarial fue esencial para terminar a tiempo esta cuarta edición.

La editora fue Betsy Jones. El editor de desarrollo senior fue Kelly Butcher. El director de proyecto fue Jill Peter. El director de marketing ejecutivo fue John Wannemacher. El ilustrador de la portada fue Paul Tumbaugh mientras que el diseñador de la portada fue JoAnne Schopler. El editor de copia fue George Watson. El corrector de pruebas fue Marie Zartman. El productor del material complementario fue Jodi Banowetz. El diseñador fue Rick Noel. El indexador fue Tobiah Waldron.

Esta edición está basada en las tres ediciones previas, así que los autores dan las gracias una vez más a las muchas personas que ayudaron con las tres primeras ediciones, incluyendo a R.B. Abhyankar, Don Batory, Haran Boral, Paul Bourgeois, Robert Brazile, Michael Carey, J. Edwards, Christos Faloutsos, Homma Farian, Alan Fekete, Shashi Gadia, Jim Gray, Le Gruenwald, Yannis Ioannidis, Hyoung-Joo Kim, Henry Korth (padre de Henry F.), Carol Kroll, Gary Lindstrom, Dave Maier, Keith Marzullo, Fletcher Mattox, Alberto Mendelzon, Héctor García-Molina, Ami Motro, Anil Nigam, Cyril Orji, Bruce Porter, Jim Peterson, K.V. Raghavan, Mark Roth, Marek Rusinkiewicz, S. Seshadri, Shashi Shekhar, Amit Sheth, Nandit Soparkar, Greg Speegle y Marianne Winslett. Lyn Dupré editó y corrigió la tercera edición del libro y Sara Strandtman editó el texto de la tercera edición. Greg Speegle, Dawn Bezviner y K. V. Raghavan nos ayudaron a preparar el manual del profesor para ediciones anteriores. La nueva portada es una evolución de las portadas de las tres primeras ediciones. Marilyn Turnamian creó un primer boceto del diseño de la portada para esta edición. La idea de usar barcos como parte del concepto de la portada fue sugerida originalmente por Bruce Stephan.

Finalmente, Sudarshan desearía agradecer a su esposa, Sita, por su amor y apoyo, a su hijo de dos años Madhur por su amor, y a su madre, Indira, por su apoyo. Hank desearía agradecer a su esposa, Joan, y a sus hijos, Abby y Joe, por su amor y comprensión. Avi desearía agradecer a su esposa Haya y a su hijo Aaron por su paciencia y apoyo durante la revisión de este libro.

A.S.
H.F.K.
S.S.

CAPÍTULO

1 INTRODUCCIÓN

UN **sistema gestor de bases de datos** (SGBD) consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos datos. La colección de datos, normalmente denominada **base de datos**, contiene información relevante para una empresa. El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto *práctica* como *eficiente*.

Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la información. Además, los sistemas de bases de datos deben proporcionar la fiabilidad de la información almacenada, a pesar de las caídas del sistema o los intentos de acceso sin autorización. Si los datos van a ser compartidos entre diversos usuarios, el sistema debe evitar posibles resultados anómalos.

Dado que la información es tan importante en la mayoría de las organizaciones, los científicos informáticos han desarrollado un amplio conjunto de conceptos y técnicas para la gestión de los datos. En este capítulo se presenta una breve introducción a los principios de los sistemas de bases de datos.

1.1. APLICACIONES DE LOS SISTEMAS DE BASES DE DATOS

Las bases de datos son ampliamente usadas. Las siguientes son algunas de sus aplicaciones más representativas:

- *Banca*. Para información de los clientes, cuentas y préstamos, y transacciones bancarias.
- *Líneas aéreas*. Para reservas e información de planificación. Las líneas aéreas fueron de los primeros en usar las bases de datos de forma distribuida geográficamente (los terminales situados en todo el mundo accedían al sistema de bases de datos centralizado a través de las líneas telefónicas y otras redes de datos).
- *Universidades*. Para información de los estudiantes, matrículas de las asignaturas y cursos.
- *Transacciones de tarjetas de crédito*. Para compras con tarjeta de crédito y generación mensual de extractos.
- *Telecomunicaciones*. Para guardar un registro de las llamadas realizadas, generación mensual de facturas, manteniendo el saldo de las tarjetas telefónicas de prepago y para almacenar información sobre las redes de comunicaciones.
- *Finanzas*. Para almacenar información sobre grandes empresas, ventas y compras de documentos formales financieros, como bolsa y bonos.
- *Ventas*. Para información de clientes, productos y compras.
- *Producción*. Para la gestión de la cadena de producción y para el seguimiento de la producción de elementos en las factorías, inventarios de elementos en almacenes y pedidos de elementos.
- *Recursos humanos*. Para información sobre los empleados, salarios, impuestos y beneficios, y para la generación de las nóminas.

Como esta lista ilustra, las bases de datos forman una parte esencial de casi todas las empresas actuales.

A lo largo de las últimas cuatro décadas del siglo veinte, el uso de las bases de datos creció en todas las empresas. En los primeros días, muy pocas personas interactuaron directamente con los sistemas de bases de datos, aunque sin darse cuenta interactuaron con bases de datos indirectamente (con los informes impresos como extractos de tarjetas de crédito, o mediante agentes como cajeros de bancos y agentes de reserva de líneas aéreas). Después vinieron los cajeros automáticos y permitieron a los usuarios interactuar con las bases de datos. Las interfaces telefónicas con los computadores (sistemas de respuesta vocal interactiva) también permitieron a los usuarios manejar directamente las bases de datos. Un llamador podía marcar un número y pulsar teclas del teléfono para introducir información o para seleccionar opciones alternativas, para determinar las horas de llegada o salida, por ejemplo, o para matricularse de asignaturas en una universidad.

La revolución de Internet a finales de la década de 1990 aumentó significativamente el acceso directo del

usuario a las bases de datos. Las organizaciones convirtieron muchas de sus interfaces telefónicas a las bases de datos en interfaces Web, y pusieron disponibles en línea muchos servicios. Por ejemplo, cuando se accede a una tienda de libros en línea y se busca un libro o una colección de música se está accediendo a datos almacenados en una base de datos. Cuando se solicita un pedido en línea, el pedido se almacena en una base de datos. Cuando se accede a un banco en un sitio Web y se consulta el estado de la cuenta y los movimientos, la información se recupera del sistema de bases de datos del banco. Cuando se accede a un sitio Web, la información personal puede ser recuperada de una base de datos para seleccionar los anuncios que se deberían mostrar. Más aún, los datos sobre los accesos Web pueden ser almacenados en una base de datos.

Así, aunque las interfaces de datos ocultan detalles del acceso a las bases de datos, y la mayoría de la gente ni siquiera es consciente de que están interactuando con una base de datos, el acceso a las bases de datos forma una parte esencial de la vida de casi todas las personas actualmente.

La importancia de los sistemas de bases de datos se puede juzgar de otra forma: actualmente, los vendedores de sistemas de bases de datos como Oracle están entre las mayores compañías software en el mundo, y los sistemas de bases de datos forman una parte importante de la línea de productos de compañías más diversificadas, como Microsoft e IBM.

1.2. SISTEMAS DE BASES DE DATOS FRENTE A SISTEMAS DE ARCHIVOS

Considérese parte de una empresa de cajas de ahorros que mantiene información acerca de todos los clientes y cuentas de ahorros. Una manera de mantener la información en un computador es almacenarla en archivos del sistema operativo. Para permitir a los usuarios manipular la información, el sistema tiene un número de programas de aplicación que manipula los archivos, incluyendo:

- Un programa para efectuar cargos o abonos en una cuenta.
- Un programa para añadir una cuenta nueva.
- Un programa para calcular el saldo de una cuenta.
- Un programa para generar las operaciones mensuales.

Estos programas de aplicación se han escrito por programadores de sistemas en respuesta a las necesidades de la organización bancaria.

Si las necesidades se incrementan, se añaden nuevos programas de aplicación al sistema. Por ejemplo, supóngase que las regulaciones de un nuevo gobierno permiten a las cajas de ahorros ofrecer cuentas corrientes. Como resultado se crean nuevos archivos permanentes que contengan información acerca de todas las cuentas corrientes mantenidas por el banco, y puede ser necesario escribir nuevos programas de aplicación para tratar situaciones que no existían en las cuentas de ahorro, tales como manejar descubiertos. Así, sobre la marcha, se añaden más archivos y programas de aplicación al sistema.

Este **sistema de procesamiento de archivos** típico que se acaba de describir se mantiene mediante un sistema operativo convencional. Los registros permanentes son almacenados en varios archivos y se escriben diferentes programas de aplicación para extraer registros y para añadir registros a los archivos adecuados. Antes de la llegada de los sistemas de gestión de bases de datos (SGBDs), las organizaciones normalmente han almacenado la información usando tales sistemas.

Mantener información de la organización en un sistema de procesamiento de archivos tiene una serie de inconvenientes importantes:

- **Redundancia e inconsistencia de datos.** Debido a que los archivos y programas de aplicación son creados por diferentes programadores en un largo período de tiempo, los diversos archivos tienen probablemente diferentes formatos y los programas pueden estar escritos en diferentes lenguajes. Más aún, la misma información puede estar duplicada en diferentes lugares (archivos). Por ejemplo, la dirección y número de teléfono de un cliente particular puede aparecer en un archivo que contenga registros de cuentas de ahorros y en un archivo que contenga registros de una cuenta corriente. Esta redundancia conduce a un almacenamiento y coste de acceso más altos. Además, puede conducir a **inconsistencia de datos;** es decir, las diversas copias de los mismos datos pueden no coincidir. Por ejemplo, un cambio en la dirección del cliente puede estar reflejado en los registros de las cuentas de ahorro pero no estarlo en el resto del sistema.
- **Dificultad en el acceso a los datos.** Supóngase que uno de los empleados del banco necesita averiguar los nombres de todos los clientes que viven en el distrito postal 28733 de la ciudad. El empleado pide al departamento de procesamiento de datos que genere dicha lista. Debido a que esta petición no fue prevista cuando el sistema original fue diseñado, no hay un programa de aplicación a mano para satisfacerla. Hay, sin embargo, un programa de aplicación que genera la lista de *todos* los clientes. El empleado del banco tiene ahora dos opciones: bien obtener la lista de todos los clientes y obtener la información que necesita manualmente, o bien pedir al departamento de procesamiento de datos que haga

que un programador de sistemas escriba el programa de aplicación necesario. Ambas alternativas son obviamente insatisfactorias. Supóngase que se escribe tal programa y que, varios días más tarde, el mismo empleado necesita arreglar esa lista para incluir sólo aquellos clientes que tienen una cuenta con saldo de 10.000 € o más. Como se puede esperar, un programa para generar tal lista no existe. De nuevo, el empleado tiene que elegir entre dos opciones, ninguna de las cuales es satisfactoria.

La cuestión aquí es que el entorno de procesamiento de archivos convencional no permite que los datos necesarios sean obtenidos de una forma práctica y eficiente. Se deben desarrollar sistemas de recuperación de datos más interesantes para un uso general.

- **Aislamiento de datos.** Debido a que los datos están dispersos en varios archivos, y los archivos pueden estar en diferentes formatos, es difícil escribir nuevos programas de aplicación para recuperar los datos apropiados.
- **Problemas de integridad.** Los valores de los datos almacenados en la base de datos deben satisfacer ciertos tipos de **restricciones de consistencia.** Por ejemplo, el saldo de una cuenta bancaria no puede nunca ser más bajo de una cantidad predeterminada (por ejemplo 25 €). Los desarrolladores hacen cumplir esas restricciones en el sistema añadiendo el código apropiado en los diversos programas de aplicación. Sin embargo, cuando se añaden nuevas restricciones, es difícil cambiar los programas para hacer que se cumplan. El problema es complicado cuando las restricciones implican diferentes elementos de datos de diferentes archivos.
- **Problemas de atomicidad.** Un sistema de un computador, como cualquier otro dispositivo mecánico o eléctrico, está sujeto a fallo. En muchas aplicaciones es crucial asegurar que, una vez que un fallo ha ocurrido y se ha detectado, los datos se restauran al estado de consistencia que existía antes del fallo. Consideremos un programa para transferir 50 € desde la cuenta A a la B. Si ocurre un fallo del sistema durante la ejecución del programa, es posible que los 50 € fueron eliminados de la cuenta A pero no abonados a la cuenta B, resultando un estado de la base de datos inconsistente. Claramente, es esencial para la consistencia de la base de datos que ambos, el abono y el cargo tengan lugar, o que ninguno tenga lugar. Es decir, la transferencia de fondos debe ser *atómica*: ésta debe ocurrir en ellos por completo o no ocurrir en absoluto. Es difícil asegurar esta propiedad en un sistema de procesamiento de archivos convencional.
- **Anomalías en el acceso concurrente.** Conforme se ha ido mejorando el conjunto de ejecución de los sistemas y ha sido posible una respuesta en tiempo más rápida, muchos sistemas han ido permitiendo a múltiples usuarios actualizar los datos simultáneamente. En tales sistemas un entorno de interacción de actualizaciones concurrentes puede dar lugar a datos inconsistentes. Considérese una cuenta bancaria A, que contiene 500 €. Si dos clientes retiran fondos (por ejemplo 50 € y 100 € respectivamente) de la cuenta A en aproximadamente el mismo tiempo, el resultado de las ejecuciones concurrentes puede dejar la cuenta en un estado incorrecto (o inconsistente). Supongamos que los programas se ejecutan para cada retirada y escriben el resultado después. Si los dos programas funcionan concurrentemente, pueden leer ambos el valor 500 €, y escribir después 450 € y 400 €, respectivamente. Dependiendo de cuál escriba el último valor, la cuenta puede contener bien 450 € o bien 400 €, en lugar del valor correcto, 350 €. Para protegerse contra esta posibilidad, el sistema debe mantener alguna forma de supervisión. Sin embargo, ya que se puede acceder a los datos desde muchos programas de aplicación diferentes que no han sido previamente coordinados, la supervisión es difícil de proporcionar.
- **Problemas de seguridad.** No todos los usuarios de un sistema de bases de datos deberían poder acceder a todos los datos. Por ejemplo, en un sistema bancario, el personal de nóminas necesita ver sólo esa parte de la base de datos que tiene información acerca de varios empleados del banco. No necesitan acceder a la información acerca de las cuentas de clientes. Como los programas de aplicación se añaden al sistema de una forma ad hoc, es difícil garantizar tales restricciones de seguridad.

Estas dificultades, entre otras, han motivado el desarrollo de los sistemas de bases de datos. En este libro se verán los conceptos y algoritmos que han sido incluidos en los sistemas de bases de datos para resolver los problemas mencionados anteriormente. En la mayor parte de este libro se usa una empresa bancaria como el ejemplo de una aplicación corriente de procesamiento de datos típica encontrada en una empresa.

1.3. VISIÓN DE LOS DATOS

Un sistema de bases de datos es una colección de archivos interrelacionados y un conjunto de programas que permitan a los usuarios acceder y modificar estos archivos. Uno de los propósitos principales de un sistema de bases de datos es proporcionar a los usuarios una visión *abstracta* de los datos. Es decir, el sistema esconde ciertos detalles de cómo se almacenan y mantienen los datos.

1.3.1. Abstracción de datos

Para que el sistema sea útil debe recuperar los datos eficientemente. Esta preocupación ha conducido al diseño de estructuras de datos complejas para la representación de los datos en la base de datos. Como muchos usuarios de sistemas de bases de datos no están familiarizados con computadores, los desarrolladores esconden la complejidad a los usuarios a través de varios niveles de abstracción para simplificar la interacción de los usuarios con el sistema:

- **Nivel físico:** El nivel más bajo de abstracción describe *cómo* se almacenan realmente los datos. En el nivel físico se describen en detalle las estructuras de datos complejas de bajo nivel.
- **Nivel lógico:** El siguiente nivel más alto de abstracción describe *qué* datos se almacenan en la base de datos y qué relaciones existen entre esos datos. La base de datos completa se describe así en términos de un número pequeño de estructuras relativamente simples. Aunque la implementación de estructuras simples en el nivel lógico puede involucrar estructuras complejas del nivel físico, los usuarios del nivel lógico no necesitan preocuparse de esta complejidad. Los administradores de bases de datos, que deben decidir la información que se mantiene en la base de datos, usan el nivel lógico de abstracción.
- **Nivel de vistas:** El nivel más alto de abstracción describe sólo parte de la base de datos completa. A pesar del uso de estructuras más simples en el nivel lógico, queda algo de complejidad, debido a la variedad de información almacenada en una gran base de datos. Muchos usuarios del sistema de base de datos no necesitan toda esta información. En su lugar, tales usuarios necesitan acceder sólo a una parte de la base de datos. Para que su interacción con el sistema se simplifique, se define la abstracción del nivel de vistas. El sistema puede proporcionar muchas vistas para la misma base de datos.

La Figura 1.1 muestra la relación entre los tres niveles de abstracción.

Una analogía con el concepto de tipos de datos en lenguajes de programación puede clarificar la distinción entre los niveles de abstracción. La mayoría de lenguajes de programación de alto nivel soportan la estructura de tipo registro. Por ejemplo, en un lenguaje tipo Pascal, se pueden declarar registros como sigue:

```
type cliente = record
                 nombre-cliente : string;
                 id-cliente : string;
                 calle-cliente : string;
                 ciudad-cliente : string;
               end;
```

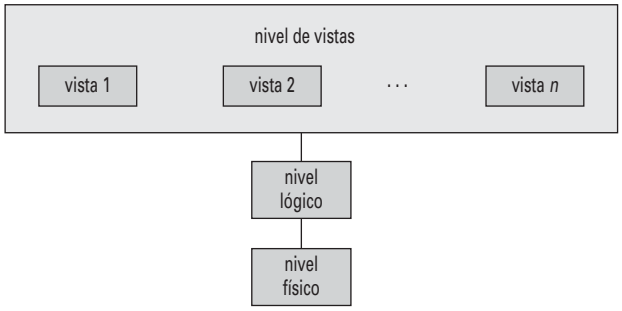

FIGURA 1.1. Los tres niveles de abstracción de datos.

Este código define un nuevo registro llamado *cliente* con cuatro campos. Cada campo tiene un nombre y un tipo asociado a él. Una empresa bancaria puede tener varios tipos de registros, incluyendo

- *cuenta*, con campos *número-cuenta* y *saldo*
- *empleado*, con campos *nombre-empleado* y *sueldo*

En el nivel físico, un registro *cliente*, *cuenta* o *empleado* se puede describir como un bloque de posiciones almacenadas consecutivamente (por ejemplo, palabras o bytes). El compilador del lenguaje esconde este nivel de detalle a los programadores. Análogamente, el sistema de base de datos esconde muchos de los detalles de almacenamiento de nivel inferior a los programadores de bases de datos. Los administradores de bases de datos pueden ser conscientes de ciertos detalles de la organización física de los datos.

En el nivel lógico cada registro de este tipo se describe mediante una definición de tipo, como se ha ilustrado en el fragmento de código previo, y se define la relación entre estos tipos de registros. Los programadores, cuando usan un lenguaje de programación, trabajan en este nivel de abstracción. De forma similar, los administradores de bases de datos trabajan habitualmente en este nivel de abstracción.

Finalmente, en el nivel de vistas, los usuarios de computadores ven un conjunto de programas de aplicación que esconden los detalles de los tipos de datos. Análogamente, en el nivel de vistas se definen varias vistas de una base de datos y los usuarios de la misma ven única y exclusivamente esas vistas. Además de esconder detalles del nivel lógico de la base de datos, las vistas también proporcionan un mecanismo de seguridad para evitar que los usuarios accedan a ciertas partes de la base de datos. Por ejemplo, los cajeros de un banco ven únicamente la parte de la base de datos que tiene información de cuentas de clientes; no pueden acceder a la información referente a los sueldos de los empleados.

1.3.2. Ejemplares y esquemas

Las bases de datos van cambiando a lo largo del tiempo conforme la información se inserta y borra. La colección de información almacenada en la base de datos en

un momento particular se denomina un *ejemplar* de la base de datos. El diseño completo de la base de datos se llama el *esquema* de la base de datos. Los esquemas son raramente modificados, si es que lo son alguna vez.

El concepto de esquemas y ejemplares de bases de datos se puede entender por analogía con un programa escrito en un lenguaje de programación. Un *esquema* de base de datos corresponde a las declaraciones de variables (junto con definiciones de tipos asociadas) en un programa. Cada variable tiene un valor particular en un instante de tiempo. Los valores de las variables en un programa en un instante de tiempo corresponde a un *ejemplar* de un esquema de bases de datos.

Los sistemas de bases de datos tiene varios esquemas divididos de acuerdo a los niveles de abstracción que se han discutido. El **esquema físico** describe el diseño físico en el nivel físico, mientras que el **esquema lógico** describe el diseño de la base de datos en el nivel lógico. Una base de datos puede tener también varios esquemas en el nivel de vistas, a menudo denominados **subesquemas**, que describen diferentes vistas de la base de datos.

De éstos, el esquema lógico es con mucho el más importante, en términos de su efecto en los programas de aplicación, ya que los programadores construyen las aplicaciones usando el esquema lógico. El esquema físico está oculto bajo el esquema lógico, y puede ser fácilmente cambiado usualmente sin afectar a los programas de aplicación. Los programas de aplicación se dice que muestran independencia física de datos si no dependen del esquema físico y, por tanto, no deben ser modificados si cambia el esquema físico.

Se estudiarán los lenguajes para la descripción de los esquemas, después de introducir la noción de modelos de datos en el siguiente apartado.

1.4. MODELOS DE LOS DATOS

Bajo la estructura de la base de datos se encuentra el **modelo de datos**: una colección de herramientas conceptuales para describir los datos, las relaciones, la semántica y las restricciones de consistencia. Para ilustrar el concepto de un modelo de datos, describimos dos modelos de datos en este apartado: el modelo entidad-relación y el modelo relacional. Los diferentes modelos de datos que se han propuesto se clasifican en tres grupos diferentes: modelos lógicos basados en objetos, modelos lógicos basados en registros y modelos físicos.

1.4.1. Modelo entidad-relación

El modelo de datos entidad-relación (E-R) está basado en una percepción del mundo real que consta de una colección de objetos básicos, llamados *entidades*, y de *relaciones* entre estos objetos. Una entidad es una «cosa» u «objeto» en el mundo real que es distinguible de otros objetos. Por ejemplo, cada persona es una entidad, y las cuentas bancarias pueden ser consideradas entidades.

Las entidades se describen en una base de datos mediante un conjunto de **atributos**. Por ejemplo, los atributos *número-cuenta* y *saldo* describen una cuenta particular de un banco y pueden ser atributos del conjunto de entidades *cuenta*. Análogamente, los atributos *nombre-cliente*, *calle-cliente* y *ciudad-cliente* pueden describir una entidad *cliente*.

Un atributo extra, *id-cliente*, se usa para identificar unívocamente a los clientes (dado que puede ser posible que haya dos clientes con el mismo nombre, dirección y ciudad. Se debe asignar un identificador único de cliente a cada cliente. En los Estados Unidos, muchas empresas utilizan el número de la seguridad social de una persona (un número único que el Gobierno de los Estados Unidos asigna a cada persona en los Estados Unidos) como identificador de cliente*.

Una **relación** es una asociación entre varias entidades. Por ejemplo, una relación *impositor* asocia un cliente con cada cuenta que tiene. El conjunto de todas las entidades del mismo tipo, y el conjunto de todas las relaciones del mismo tipo, se denominan respectivamente **conjunto de entidades** y **conjunto de relaciones**.

La estructura lógica general de una base de datos se puede expresar gráficamente mediante un *diagrama* E-R, que consta de los siguientes componentes:

- **Rectángulos**, que representan conjuntos de entidades.
- **Elipses**, que representan atributos.
- **Rombos**, que representan relaciones entre conjuntos de entidades.
- **Líneas**, que unen los atributos con los conjuntos de entidades y los conjuntos de entidades con las relaciones.

Cada componente se etiqueta con la entidad o relación que representa.

Como ilustración, considérese parte de una base de datos de un sistema bancario consistente en clientes y cuentas que tienen esos clientes. En la Figura 1.2 se

* N. del T. En España, muchas empresas usan el D.N.I. como identificador unívoco, pero a veces encuentran problemas con los números de D.N.I. que por desgracia aparecen repetidos. Para resolverlo, o bien se usa otro identificador propio de la empresa o se añade un código al número de D.N.I.

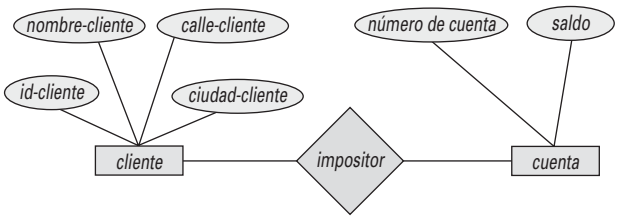

FIGURA 1.2. Ejemplo de diagrama E-R.

muestra el diagrama E-R correspondiente. El diagrama E-R indica que hay dos conjuntos de entidades *cliente* y *cuenta*, con los atributos descritos anteriormente. El diagrama también muestra la relación *impositor* entre cliente y cuenta.

Además de entidades y relaciones, el modelo E-R representa ciertas restricciones que los contenidos de la base de datos deben cumplir. Una restricción importante es la *correspondencia de cardinalidades*, que expresa el número de entidades con las que otra entidad se puede asociar a través de un conjunto de relaciones. Por ejemplo, si cada cuenta puede pertenecer sólo a un cliente, el modelo puede expresar esta restricción.

El modelo entidad-relación se utiliza habitualmente en el proceso de diseño de bases de datos, y se estudiará en produndidad en el Capítulo 2.

1.4.2. Modelo relacional

En el modelo relacional se utiliza un grupo de tablas para representar los datos y las relaciones entre ellos. Cada tabla está compuesta por varias columnas, y cada columna tiene un nombre único. En la Figura 1.3 se presenta un ejemplo de base de datos relacional consistente en tres tablas: la primera muestra los clientes de un banco, la segunda, las cuentas, y la tercera, las cuentas que pertenecen a cada cliente.

id-cliente	*nombre-cliente*	*calle-cliente*	*ciudad-cliente*
19.283.746	González	Arenal	La Granja
01.928.374	Gómez	Carretas	Cerceda
67.789.901	López	Mayor	Peguerinos
18.273.609	Abril	Preciados	Valsaín
32.112.312	Santos	Mayor	Peguerinos
33.666.999	Rupérez	Ramblas	León
01.928.374	Gómez	Carretas	Cerceda

(a) La tabla *cliente*

número-cuenta	*saldo*
C-101	500
C-215	700
C-102	400
C-305	350
C-201	900
C-217	750
C-222	700

(b) La tabla *cuenta*

id-cliente	*número-cuenta*
19.283.746	C-101
19.283.746	C-201
01.928.374	C-215
67.789.901	C-102
18.273.609	C-305
32.112.312	C-217
33.666.999	C-222
01.928.374	C-201

(b) La tabla *impositor*

FIGURA 1.3. Ejemplo de base de datos relacional.

La primera tabla, la tabla *cliente*, muestra, por ejemplo, que el cliente cuyo identificador es 19.283.746 se llama González y vive en la calle Arenal sita en La Granja. La segunda tabla, *cuenta*, muestra que las cuentas C-101 tienen un saldo de 500 € y la C-201 un saldo de 900 € respectivamente.

La tercera tabla muestra las cuentas que pertenecen a cada cliente. Por ejemplo, la cuenta C-101 pertenece al cliente cuyo identificador es 19.283.746 (González), y los clientes 19.283.746 (González) y 01.928.374 (Gómez) comparten el número de cuenta A-201 (pueden compartir un negocio).

El modelo relacional es un ejemplo de un modelo basado en registros. Los modelos basados en registros se denominan así porque la base de datos se estructura en registros de formato fijo de varios tipos. Cada tabla contiene registros de un tipo particular. Cada tipo de registro define un número fijo de campos, o atributos. Las columnas de la tabla corresponden a los atributos del tipo de registro.

No es difícil ver cómo se pueden almacenar las tablas en archivos. Por ejemplo, un carácter especial (como una coma) se puede usar para delimitar los diferentes atributos de un registro, y otro carácter especial (como un carácter de nueva línea) se puede usar para delimitar registros. El modelo relacional oculta tales detalles de implementación de bajo nivel a los desarrolladores de bases de datos y usuarios.

El modelo de datos relacional es el modelo de datos más ampliamente usado, y una amplia mayoría de sistemas de bases de datos actuales se basan en el modelo relacional. Los Capítulos 3 a 7 tratan el modelo relacional en detalle.

El modelo relacional se encuentra a un nivel de abstracción inferior al modelo de datos E-R. Los diseños de bases de datos a menudo se realizan en el modelo E-R, y después se traducen al modelo relacional; el Capítulo 2 describe el proceso de traducción. Por ejemplo, es fácil ver que las tablas *cliente* y *cuenta* corresponden a los conjuntos de entidades del mismo nombre, mientras que la tabla *impositor* corresponde al conjunto de relaciones *impositor*.

Nótese también que es posible crear esquemas en el modelo relacional que tengan problemas tales como información duplicada innecesariamente. Por ejemplo, supongamos que se almacena *número-cuenta* como un atributo del registro *cliente*. Entonces, para representar el hecho de que las cuentas C-101 y C-201 pertenecen ambas al cliente González (con identificador de cliente 19.283.746) sería necesario almacenar dos filas en la tabla *cliente*. Los valores de *nombre-cliente*, *calle-cliente* y *ciudad-cliente* de González estarían innecesariamente duplicados en las dos filas. En el Capítulo 7 se estudiará cómo distinguir buenos diseños de esquema de malos diseños.

1.4.3. Otros modelos de datos

El **modelo de datos orientado a objetos** es otro modelo de datos que está recibiendo una atención creciente.

El modelo orientado a objetos se puede observar como una extensión del modelo E-R con las nociones de encapsulación, métodos (funciones) e identidad de objeto. El Capítulo 8 examina el modelo de datos orientado a objetos.

El **modelo de datos relacional orientado a objetos** combina las características del modelo de datos orientado a objetos y el modelo de datos relacional. El Capítulo 9 lo examina.

Los modelos de datos semiestructurados permiten la especificación de datos donde los elementos de datos individuales del mismo tipo pueden tener diferentes conjuntos de atributos. Esto es diferente de los modelos de datos mencionados anteriormente, en los que cada elemento de datos de un tipo particular debe tener el mismo conjunto de atributos. El **lenguaje de marcas extensible (XML, eXtensible Markup Language)** se usa ampliamente para representar datos semiestructurados. El Capítulo 10 lo trata.

Históricamente, otros dos modelos de datos, el **modelo de datos de red** y el **modelo de datos jerárquico**, precedieron al modelo de datos relacional. Estos modelos estuvieron ligados fuertemente a la implementación subyacente y complicaban la tarea del modelado de datos. Como resultado se usan muy poco actualmente, excepto en el código de bases de datos antiguo que aún está en servicio en algunos lugares. Se describen en los apéndices A y B para los lectores interesados.

1.5. LENGUAJES DE BASES DE DATOS

Un sistema de bases de datos proporciona un **lenguaje de definición de datos** para especificar el esquema de la base de datos y un **lenguaje de manipulación de datos** para expresar las consultas a la base de datos y las modificaciones. En la práctica, los lenguajes de definición y manipulación de datos no son dos lenguajes separados; en su lugar simplemente forman partes de un único lenguaje de bases de datos, tal como SQL, ampliamente usado.

1.5.1. Lenguaje de definición de datos

Un esquema de base de datos se especifica mediante un conjunto de definiciones expresadas mediante un lenguaje especial llamado **lenguaje de definición de datos (LDD)**.

Por ejemplo, la siguiente instrucción en el lenguaje SQL define la tabla *cuenta*:

create table *cuenta*
(*número-cuenta* **char**(10),
saldo **integer**)

La ejecución de la instrucción LDD anterior crea la tabla *cuenta*. Además, actualiza un conjunto especial de tablas denominado **diccionario de datos** o **directorio de datos**.

Un diccionario de datos contiene **metadatos**, es decir, datos acerca de los datos. El esquema de una tabla es un ejemplo de metadatos. Un sistema de base de datos consulta el diccionario de datos antes de leer o modificar los datos reales.

Especificamos el almacenamiento y los métodos de acceso usados por el sistema de bases de datos por un conjunto de instrucciones en un tipo especial de LDD denominado lenguaje de **almacenamiento y definición de datos**. Estas instrucciones definen los detalles de implementación de los esquemas de base de datos, que se ocultan usualmente a los usuarios.

Los valores de datos almacenados en la base de datos deben satisfacer ciertas **restricciones de consistencia**. Por ejemplo, supóngase que el saldo de una cuenta no debe caer por debajo de 100 €. El LDD proporciona facilidades para especificar tales restricciones. Los sistemas de bases de datos comprueban estas restricciones cada vez que se actualiza la base de datos.

1.5.2. Lenguaje de manipulación de datos

La **manipulación de datos** es:

- La recuperación de información almacenada en la base de datos.
- La inserción de información nueva en la base de datos.
- El borrado de información de la base de datos.
- La modificación de información almacenada en la base de datos.

Un ***lenguaje de manipulación de datos* (LMD)** es un lenguaje que permite a los usuarios acceder o manipular los datos organizados mediante el modelo de datos apropiado. Hay dos tipos básicamente:

- **LMDs procedimentales.** Requieren que el usuario especifique *qué* datos se necesitan y *cómo* obtener esos datos.
- **LMDs declarativos** (también conocidos como LMDs **no procedimentales**). Requieren que el usuario especifique *qué* datos se necesitan *sin* especificar cómo obtener esos datos.

Los LMDs declarativos son más fáciles de aprender y usar que los LMDs procedimentales. Sin embargo, como el usuario no especifica cómo conseguir los datos, el sistema de bases de datos tiene que determinar un medio eficiente de acceder a los datos. El componente LMD del lenguaje SQL es no procedimental.

Una **consulta** es una instrucción de solicitud para recuperar información. La parte de un LMD que implica recuperación de información se llama **lenguaje de consultas**. Aunque técnicamente sea incorrecto, en la práctica se usan los términos *lenguaje de consultas* y *lenguaje de manipulación de datos* como sinónimos.

Esta consulta en el lenguaje SQL encuentra el nombre del cliente cuyo identificador de cliente es 19.283.746:

```
select cliente.nombre-cliente
from cliente
where cliente.id-cliente = '19 283 746'
```

La consulta especifica que las filas *de* (from) la tabla *cliente donde* (where) el id-cliente es 19 283 746 se debe recuperar, y que se debe mostrar el atributo *nombre-cliente* de estas filas. Si se ejecutase la consulta con la tabla de la Figura 1.3, se mostraría el nombre González.

Las consultas pueden involucrar información de más de una tabla. Por ejemplo, la siguiente consulta encuentra el saldo de todas las cuentas pertenecientes al cliente cuyo identificador de cliente es 19 283 746.

```
select cuenta.saldo
from impositor, cuenta
where impositor.id-cliente = '19-283-746' and
    impositor.número-cuenta = cuenta.número-
    cuenta
```

Si la consulta anterior se ejecutase con las tablas de la Figura 1.3, el sistema encontraría que las dos cuentas denominadas C-101 y C-201 pertenecen al cliente 19 283 746 e imprimiría los saldos de las dos cuentas, es decir, 500 y 900 €.

Hay varios lenguajes de consulta de bases de datos en uso, ya sea comercialmente o experimentalmente. Se estudiará el lenguaje de consultas más ampliamente usado, SQL, en el Capítulo 4. También se estudiarán otros lenguajes de consultas en el Capítulo 5.

Los niveles de abstracción que se discutieron en el Apartado 1.3 se aplican no solo a la definición o estructuración de datos, sino también a la manipulación de datos. En el nivel físico se deben definir algoritmos que permitan un acceso eficiente a los datos. En los niveles superiores de abstracción se enfatiza la facilidad de uso. El objetivo es proporcionar una interacción humana eficiente con el sistema. El componente procesador de consultas del sistema de bases de datos (que se estudia en los Capítulos 13 y 14) traduce las consultas LMD en secuencias de acciones en el nivel físico del sistema de bases de datos.

1.5.3. Acceso a la base de datos desde programas de aplicación

Los **programas de aplicación** son programas que se usan para interaccionar con la base de datos. Los programas de aplicación se escriben usualmente en un lenguaje *anfitrión*, tal como Cobol, C, C++ o Java. En el sistema bancario algunos ejemplos son programas que emiten los cheques de las nóminas, las cuentas de débito, las cuentas de crédito o las transferencias de fondos entre cuentas.

Para acceder a la base de datos, las instrucciones LMD necesitan ser ejecutadas desde el lenguaje anfitrión. Hay dos maneras de hacerlo:

- Proporcionando una interfaz de programas de aplicación (conjunto de procedimientos) que se pueden usar para enviar instrucciones LMD y LDD a la base de datos, y recuperar los resultados.

  El estándar de conectividad abierta de bases de datos (ODBC, Open Data Base Connectivity) definido por Microsoft para el uso con el lenguaje C es un estándar de interfaz de programas de aplicación usado comúnmente. El estándar conectividad de Java con bases de datos (JDBC, Java Data Base Connectivity) proporciona características correspondientes para el lenguaje Java.
- Extendiendo la sintaxis del lenguaje anfitrión para incorporar llamadas LMD dentro del programa del lenguaje anfitrión. Usualmente, un carácter especial precede a las llamadas LMD, y un preprocesador, denominado el **precompilador LMD**, convierte las instrucciones LMD en llamadas normales a procedimientos en el lenguaje anfitrión.

1.6. USUARIOS Y ADMINISTRADORES DE LA BASE DE DATOS

Un objetivo principal de un sistema de bases de datos es recuperar información y almacenar nueva información en la base de datos. Las personas que trabajan con una base de datos se pueden catalogar como usuarios de bases de datos o como administradores de bases de datos.

1.6.1. Usuarios de bases de datos e interfaces de usuario

Hay cuatro tipos diferentes de usuarios de un sistema de base de datos, diferenciados por la forma en que ellos esperan interactuar con el sistema. Se han diseñado diferentes tipo de interfaces de usuario para diferentes tipos de usuarios.

- **Usuarios normales.** Son usuarios no sofisticados que interactúan con el sistema mediante la invocación de alguno de los programas de aplicación permanentes que se ha escrito previamente. Por ejemplo, un cajero bancario que necesita transferir 50 € de la cuenta *A* a la cuenta *B* invoca un programa llamado *transferir*. Este programa pide al

cajero el importe de dinero a transferir, la cuenta de la que el dinero va a ser transferido y la cuenta a la que el dinero va a ser transferido.

Como otro ejemplo, considérese un usuario que desee encontrar su saldo de cuenta en World Wide Web. Tal usuario podría acceder a un formulario en el que introduce su número de cuenta. Un programa de aplicación en el servidor Web recupera entonces el saldo de la cuenta, usando el número de cuenta proporcionado, y pasa la información al usuario.

La interfaz de usuario normal para los usuarios normales es una interfaz de formularios, donde el usuario puede rellenar los campos apropiados del formulario. Los usuarios normales pueden también simplemente leer *informes* generados de la base de datos.

- **Programadores de aplicaciones.** Son profesionales informáticos que escriben programas de aplicación. Los programadores de aplicaciones pueden elegir entre muchas herramientas para desarrollar interfaces de usuario. Las herramientas de **desarrollo rápido de aplicaciones (DRA)** son herramientas que permiten al programador de aplicaciones construir formularios e informes sin escribir un programa. Hay también tipos especiales de lenguajes de programación que combinan estructuras de control imperativo (por ejemplo, para bucles for, bucles while e instrucciones if-then-else) con instrucciones del lenguaje de manipulación de datos. Estos lenguajes, llamados a veces *lenguajes de cuarta generación*, a menudo incluyen características especiales para facilitar la generación de formularios y la presentación de datos en pantalla. La mayoría de los sistemas de bases de datos comerciales incluyen un lenguaje de cuarta generación.

- **Los usuarios sofisticados** interactúan con el sistema sin programas escritos. En su lugar, ellos forman sus consultas en un lenguaje de consulta de bases de datos. Cada una de estas consultas se envía al *procesador de consultas*, cuya función es transformar instrucciones LMD a instrucciones que el gestor de almacenamiento entienda. Los analistas que envían las consultas para explorar los datos en la base de datos entran en esta categoría.

  Las herramientas de **procesamiento analítico en línea (OLAP, Online Analytical Processing)** simplifican la labor de los analistas permitiéndoles ver resúmenes de datos de formas diferentes. Por ejemplo, un analista puede ver las ventas totales por región (por ejemplo, norte, sur, este y oeste), o por producto, o por una combinación de la región y del producto (es decir, las ventas totales de cada producto en cada región). Las herramientas también permiten al analista seleccionar regiones específicas, examinar los datos con más detalle (por ejemplo, ventas por ciudad dentro de una región) o examinar los datos con menos detalle (por ejemplo, agrupando productos por categoría).

  Otra clase de herramientas para los analistas son las herramientas de **recopilación de datos**, que les ayudan a encontrar ciertas clases de patrones de datos.

  En el Capítulo 22 se estudiarán las herramientas de recopilación de datos.

- **Usuarios especializados.** Son usuarios sofisticados que escriben aplicaciones de bases de datos especializadas que no son adecuadas en el marco de procesamiento de datos tradicional. Entre estas aplicaciones están los sistemas de diseño asistido por computador, sistemas de bases de conocimientos y sistemas expertos, sistemas que almacenan los datos con tipos de datos complejos (por ejemplo, datos gráficos y datos de audio) y sistemas de modelado del entorno. Varias de estas aplicaciones se tratan en los Capítulos 8 y 9.

1.6.2. Administrador de la base de datos

Una de las principales razones de usar SGBDs es tener un control centralizado tanto de los datos como de los programas que acceden a esos datos. La persona que tiene este control central sobre el sistema se llama **administrador de la base de datos (ABD)**. Las funciones del ABD incluyen las siguientes:

- **Definición del esquema.** El ABD crea el esquema original de la base de datos escribiendo un conjunto de instrucciones de definición de datos en el LDD.
- **Definición de la estructura y del método de acceso.**
- **Modificación del esquema y de la organización física.** Los ABD realizan cambios en el esquema y en la organización física para reflejar las necesidades cambiantes de la organización, o para alterar la organización física para mejorar el rendimiento.
- **Concesión de autorización para el acceso a los datos.** La concesión de diferentes tipos de autorización permite al administrador de la base de datos determinar a qué partes de la base de datos puede acceder cada usuario. La información de autorización se mantiene en una estructura del sistema especial que el sistema de base de datos consulta cuando se intenta el acceso a los datos en el sistema.
- **Mantenimiento rutinario.** Algunos ejemplos de actividades rutinarias de mantenimiento del administrado de la base de datos son:
  - Copia de seguridad periódica de la base de datos, bien sobre cinta o sobre servidores remotos, para prevenir la pérdida de datos en caso de desastres como inundaciones.

— Asegurarse de que haya suficiente espacio libre en disco para las operaciones normales y aumentar el espacio en disco según sea necesario.

— Supervisión de los trabajos que se ejecuten en la base de datos y asegurarse de que el rendimiento no se degrada por tareas muy costosas iniciadas por algunos usuarios.

1.7. GESTIÓN DE TRANSACCIONES

Varias operaciones sobre la base de datos forman a menudo una única unidad lógica de trabajo. Un ejemplo que se vio en el Apartado 1.2 es la transferencia de fondos, en el que una cuenta (*A*) se carga y otra cuenta (*B*) se abona. Claramente es esencial que, o bien tanto el cargo como el abono tengan lugar, o bien no ocurra ninguno. Es decir, la transferencia de fondos debe ocurrir por completo o no ocurrir en absoluto. Este requisito de todo o nada se denomina **atomicidad**. Además, es esencial que la ejecución de la transferencia de fondos preserve la consistencia de la base de datos. Es decir, el valor de la suma $A + B$ se debe preservar. Este requisito de corrección se llama **consistencia**. Finalmente, tras la ejecución correcta de la transferencia de fondos, los nuevos valores de las cuentas *A* y *B* deben persistir, a pesar de la posibilidad de fallo del sistema. Este requisito de persistencia se llama **durabilidad**.

Una **transacción** es una colección de operaciones que se lleva a cabo como una única función lógica en una aplicación de bases de datos. Cada transacción es una unidad de atomicidad y consistencia. Así, se requiere que las transacciones no violen ninguna restricción de consistencia de la base de datos. Es decir, si la base de datos era consistente cuando la transacción comenzó, la base de datos debe ser consistente cuando la transacción termine con éxito. Sin embargo, durante la ejecución de una transacción, puede ser necesario permitir inconsistencias temporalmente, ya que o el cargo de *A* o el abono de *B* se debe realizar uno antes que otro. Esta inconsistencia temporal, aunque necesaria, puede conducir a dificultades si ocurre un fallo.

Es responsabilidad del programador definir adecuadamente las diferentes transacciones, de tal manera que cada una preserve la consistencia de la base de datos. Por ejemplo, la transacción para transferir fondos de la cuenta *A* a la cuenta *B* se podría definir como compuesta de dos programas separados: uno que carga la cuenta A y otro que abona la cuenta B. La ejecución de estos dos programas uno después del otro preservará realmente la consistencia. Sin embargo, cada programa en sí mismo no transforma la base de datos de un estado consistente en otro nuevo estado consistente. Así, estos programas no son transacciones.

Asegurar las propiedades de atomicidad y durabilidad es responsabilidad del propio sistema de bases de datos, específicamente del **componente de gestión de transacciones**. En ausencia de fallos, toda transacción completada con éxito y atómica se archiva fácilmente. Sin embargo, debido a diversos tipos de fallos, una transacción puede no siempre completar su ejecución con éxito. Si se asegura la propiedad de atomicidad, una transacción que falle no debe tener efecto en el estado de la base de datos. Así, la base de datos se restaura al estado en que estaba antes de que la transacción en cuestión comenzara su ejecución. El sistema de bases de datos debe realizar la **recuperación de fallos**, es decir, detectar los fallos del sistema y restaurar la base de datos al estado que existía antes de que ocurriera el fallo.

Finalmente, cuando varias transacciones actualizan la base de datos concurrentemente, la consistencia de los datos puede no ser preservada, incluso aunque cada transacción individualmente sea correcta. Es responsabilidad del **gestor de control de concurrencia** controlar la interacción entre las transacciones concurrentes para asegurar la consistencia de la base de datos.

Los sistemas de bases de datos diseñados para uso sobre pequeños computadores personales pueden no tener todas las características vistas. Por ejemplo, muchos sistemas pequeños imponen la restricción de permitir el acceso a un único usuario a la base de datos en un instante de tiempo. Otros dejan las tareas de copias de seguridad y recuperación a los usuarios. Estas restricciones permiten un gestor de datos más pequeño, con menos requisitos de recursos físicos, especialmente de memoria principal. Aunque tales enfoques de bajo coste y prestaciones son suficientes para bases de datos personales pequeñas, son inadecuadas para satisfacer las necesidades de una empresa de media a gran escala.

1.8. ESTRUCTURA DE UN SISTEMA DE BASES DE DATOS

Un sistema de bases de datos se divide en módulos que se encargan de cada una de las responsabilidades del sistema completo. Los componentes funcionales de un sistema de bases de datos se pueden dividir a grandes rasgos en los componentes gestor de almacenamiento y procesador de consultas.

El gestor de consultas es importante porque las bases de datos requieren normalmente una gran cantidad de

espacio de almacenamiento. Las bases de datos corporativas tienen un tamaño de entre cientos de gigabytes y, para las mayores bases de datos, terabytes de datos. Un gigabyte son 1.000 megabytes (1.000 millones de bytes), y un terabyte es 1 millón de megabytes (1 billón de bytes). Debido a que la memoria principal de los computadores no puede almacenar esta gran cantidad de información, esta se almacena en discos. Los datos se trasladan entre el disco de almacenamiento y la memoria principal cuando es necesario. Como la transferencia de datos a y desde el disco es lenta comparada con la velocidad de la unidad central de procesamiento, es fundamental que el sistema de base de datos estructure los datos para minimizar la necesidad de movimiento de datos entre el disco y la memoria principal.

El procesador de consultas es importante porque ayuda al sistema de bases de datos a simplificar y facilitar el acceso a los datos. Las vistas de alto nivel ayudan a conseguir este objetivo. Con ellas, los usuarios del sistema no deberían ser molestados innecesariamente con los detalles físicos de implementación del sistema. Sin embargo, el rápido procesamiento de las actualizaciones y de las consultas es importante. Es trabajo del sistema de bases de datos traducir las actualizaciones y las consultas escritas en un lenguaje no procedimental, en el nivel lógico, en una secuencia de operaciones en el nivel físico.

1.8.1. Gestor de almacenamiento

Un *gestor de almacenamiento* es un módulo de programa que proporciona la interfaz entre los datos de bajo nivel en la base de datos y los programas de aplicación y consultas emitidas al sistema. El gestor de almacenamiento es responsable de la interacción con el gestor de archivos. Los datos en bruto se almacenan en disco usando un sistema de archivos, que está disponible habitualmente en un sistema operativo convencional. El gestor de almacenamiento traduce las diferentes instrucciones LMD a órdenes de un sistema de archivos de bajo nivel. Así, el gestor de almacenamiento es responsable del almacenamiento, recuperación y actualización de los datos en la base de datos.

Los componentes del gestor de almacenamiento incluyen:

- **Gestor de autorización e integridad**, que comprueba que se satisfagan las restricciones de integridad y la autorización de los usuarios para acceder a los datos.
- **Gestor de transacciones**, que asegura que la base de datos quede en un estado consistente (correcto) a pesar de los fallos del sistema, y que las ejecuciones de transacciones concurrentes ocurran si conflictos.
- **Gestor de archivos**, que gestiona la reserva de espacio de almacenamiento de disco y las estructuras de datos usadas para representar la información almacenada en disco.
- **Gestor de memoria intermedia**, que es responsable de traer los datos del disco de almacenamiento a memoria principal y decidir qué datos tratar en memoria caché. El gestor de memoria intermedia es una parte crítica del sistema de bases de datos, ya que permite que la base de datos maneje tamaños de datos que son mucho mayores que el tamaño de la memoria principal.

El gestor de almacenamiento implementa varias estructuras de datos como parte de la implementación física del sistema:

- **Archivos de datos**, que almacenan la base de datos en sí.
- **Diccionario de datos**, que almacena metadatos acerca de la estructura de la base de datos, en particular, el esquema de la base de datos.
- **Índices**, que proporcionan acceso rápido a elementos de datos que tienen valores particulares.

1.8.2. Procesador de consultas

Los componentes del procesador de consultas incluyen:

- **Intérprete del LDD**, que interpreta las instrucciones del LDD y registra las definiciones en el diccionario de datos.
- **Compilador del LMD**, que traduce las instrucciones del LMD en un lenguaje de consultas a un plan de evaluación que consiste en instrucciones de bajo nivel que entiende el motor de evaluación de consultas.

  Una consulta se puede traducir habitualmente en varios planes de ejecución alternativos que proporcionan el mismo resultado. El compilador del LMD también realiza **optimización de consultas**, es decir, elige el plan de evaluación de menor coste de entre todas las alternativas.
- **Motor de evaluación de consultas**, que ejecuta las instrucciones de bajo nivel generadas por el compilador del LMD.

En la Figura 1.4 se muestran estos componentes y sus conexiones.

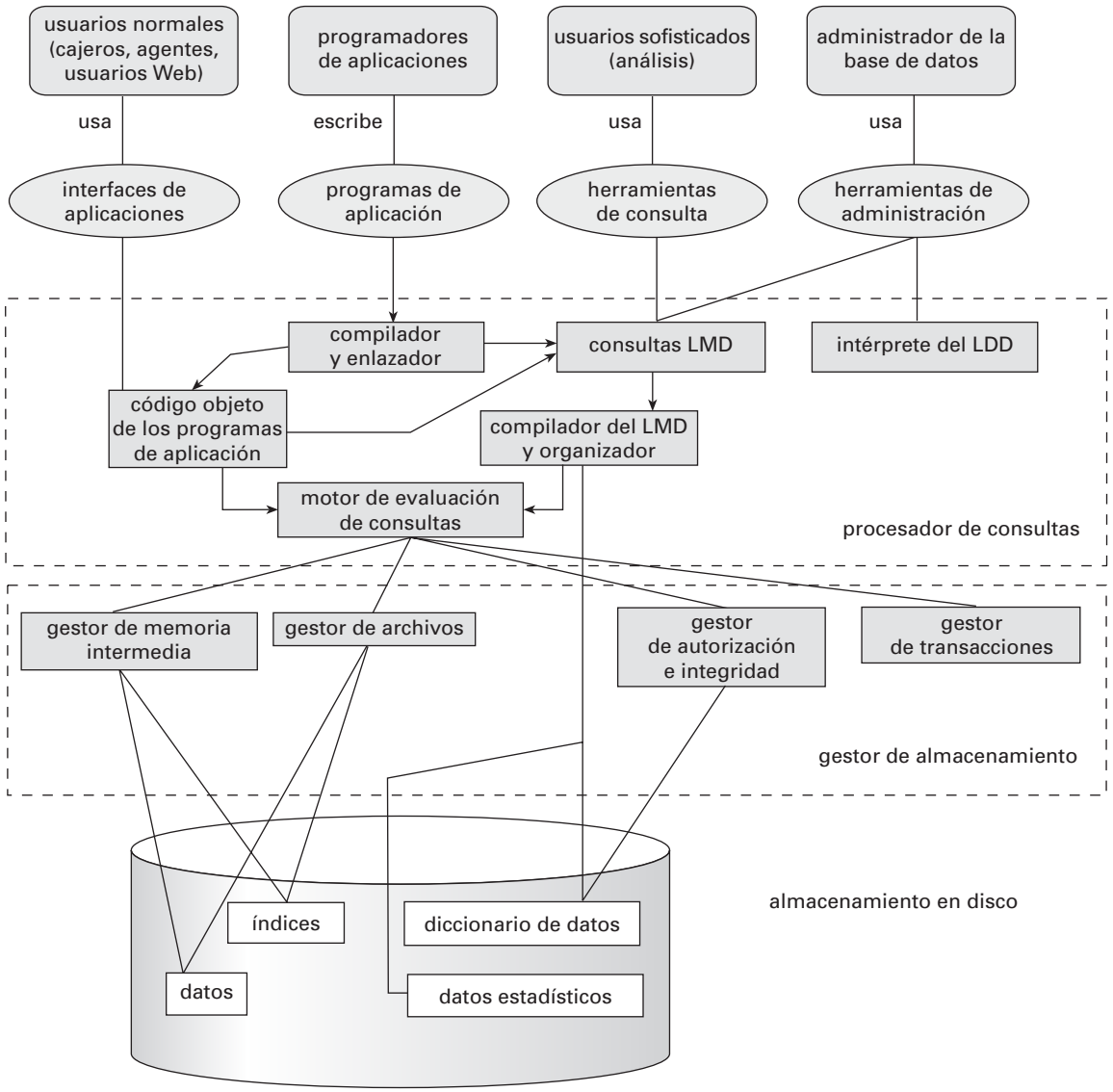

FIGURA 1.4. Estructura del sistema.

1.9. ARQUITECTURAS DE APLICACIONES

La mayoría de usuarios de un sistema de bases de datos no están situados actualmente junto al sistema de bases de datos, sino que se conectan a él a través de una red. Se puede diferenciar entonces entre las máquinas **cliente**, en donde trabajan los usuarios remotos de la base de datos, y las máquinas **servidor**, en las que se ejecuta el sistema de bases de datos.

Las aplicaciones de bases de datos se dividen usualmente en dos o tres partes, como se ilustra en la Figura 1.5. En una **arquitectura de dos capas**, la aplicación se divide en un componente que reside en la máquina cliente, que llama a la funcionalidad del sistema de bases de datos en la máquina servidor mediante instrucciones del lenguaje de consultas. Los estándares de interfaces de programas de aplicación como ODBC y JDBC se usan para la interacción entre el cliente y el servidor.

En cambio, en una **arquitectura de tres capas**, la máquina cliente actúa simplemente como frontal y no contiene ninguna llamada directa a la base de datos. En su lugar, el cliente se comunica con un **servidor de aplicaciones**, usualmente mediante una interfaz de formularios. El servidor de aplicaciones, a su vez, se comunica con el sistema de bases de datos para acceder a los datos. La **lógica de negocio** de la aplicación, que establece las acciones a realizar bajo determinadas condiciones, se incorpora en el servidor de aplicaciones, en lugar de ser distribuida a múltiples clientes. Las aplicaciones de tres capas son más apropiadas para grandes aplicaciones, y para las aplicaciones que se ejecutan en World Wide Web.

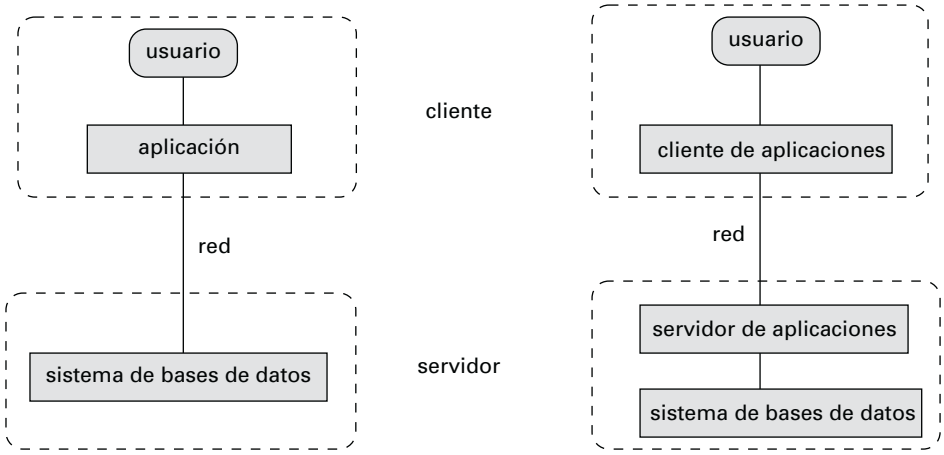

FIGURA 1.5. Arquitecturas de dos y tres capas.

1.10. HISTORIA DE LOS SISTEMAS DE BASES DE DATOS

El procesamiento de datos impulsa el crecimiento de los computadores, como ocurriera en los primeros días de los computadores comerciales. De hecho, la automatización de las tareas de procesamiento de datos precede a los computadores. Las tarjetas perforadas, inventadas por Hollerith, se usaron en los principios del siglo xx para registrar los datos del censo de los EE.UU., y se usaron sistemas mecánicos para procesar las tarjetas y para tabular los resultados. Las tarjetas perforadas posteriormente se usaron ampliamente como medio para introducir datos en los computadores.

Las técnicas del almacenamiento de datos han evolucionado a lo largo de los años:

- **Década de 1950 y principios de la década de 1960.** Se desarrollaron las cintas magnéticas para el almacenamiento de datos. Las tareas de procesamiento de datos tales como las nóminas fueron automatizadas, con los datos almacenados en cintas. El procesamiento de datos consistía en leer datos de una o más cintas y escribir datos en una nueva cinta. Los datos también se podían introducir desde paquetes de tarjetas perforadas e impresos en impresoras. Por ejemplo, los aumentos de sueldo se procesaban introduciendo los aumentos en las tarjetas perforadas y leyendo el paquete de cintas perforadas en sincronización con una cinta que contenía los detalles maestros de los salarios. Los registros debían estar igualmente ordenados. Los aumentos de sueldo tenían que añadirse a los sueldos leídos de la cinta maestra, y escribirse en una nueva cinta; esta nueva cinta se convertía en la nueva cinta maestra.

  Las cintas (y los paquetes de tarjetas perforadas) sólo se podían leer secuencialmente, y los tamaños de datos eran mucho mayores que la memoria principal; así, los programas de procesamiento de datos tenían que procesar los datos según un determinado orden, leyendo y mezclando datos de cintas y paquetes de tarjetas perforadas.
- **Finales de la década de 1960 y la década de 1970.** El amplio uso de los discos fijos a finales de la década de 1960 cambió en gran medida el escenario del procesamiento de datos, ya que los discos fijos permitieron el acceso directo a los datos. La ubicación de los datos en disco no era importante, ya que a cualquier posición del disco se podía acceder en sólo decenas de milisegundo. Los datos se liberaron de la tiranía de la secuencialidad. Con los discos pudieron desarrollarse las bases de datos de red y jerárquicas, que permitieron que las estructuras de datos tales como listas y árboles pudieran almacenarse en disco. Los programadores pudieron construir y manipular estas estructuras de datos.

  Un artículo histórico de Codd [1970] definió el modelo relacional y formas no procedimentales de consultar los datos en el modelo relacional, y nacieron las bases de datos relacionales. La simplicidad del modelo relacional y la posibilidad de ocultar completamente los detalles de implementación al programador fueron realmente atractivas. Codd obtuvo posteriormente el prestigioso premio Turing de la ACM (Association of Computing Machinery, asociación de maquinaria informática) por su trabajo.
- **Década de 1980.** Aunque académicamente interesante, el modelo relacional no se usó inicialmente en la práctica debido a sus inconvenientes por el rendimiento; las bases de datos relacionales no pudieron competir con el rendimiento de las bases

de datos de red y jerárquicas existentes. Esta situación cambió con System R, un proyecto innovador en IBM Research que desarrolló técnicas para la construcción de un sistema de bases de datos relacionales eficiente. En Astrahan et al. [1976] y Chamberlin et al. [1981] se pueden encontrar excelentes visiones generales de System R. El prototipo de System R completamente funcional condujo al primer producto de bases de datos relacionales de IBM: SQL/DS. Los primeros sistemas de bases de datos relacionales, como DB2 de IBM, Oracle, Ingres y Rdb de DEC, jugaron un importante papel en el desarrollo de técnicas para el procesamiento eficiente de consultas declarativas. En los principios de la década de 1980 las bases de datos relacionales llegaron a competir con los sistemas de bases de datos jerárquicas y de red incluso en el área de rendimiento. Las bases de datos relacionales fueron tan sencillas de usar que finalmente reemplazaron a las bases de datos jerárquicas y de red; los programadores que usaban estas bases de datos estaban forzados a tratar muchos detalles de implementación de bajo nivel y tenían que codificar sus consultas de forma procedimental. Aún más importante, debían tener presente el rendimiento durante el diseño de sus programas, lo que implicaba un gran esfuerzo. En cambio, en una base de datos relacional, casi todas estas tareas de bajo nivel se realizan automáticamente por la base de datos, liberando al programador en el nivel lógico. Desde su escalada en el dominio en la década de 1980, el modelo relacional ha conseguido el reinado supremo entre todos los modelos de datos.

La década de 1980 también fue testigo de una gran investigación en las bases de datos paralelas y distribuidas, así como del trabajo inicial en las bases de datos orientadas a objetos.

- **Principios de la década de 1990.** El lenguaje SQL se diseñó fundamentalmente para las aplicaciones de ayuda a la toma de decisiones, que son intensivas en consultas, mientras que el objetivo principal de las bases de datos en la década de 1980 fue las aplicaciones de procesamiento de transacciones, que son intensivas en actualizaciones. La ayuda a la toma de decisiones y las consultas reemergieron como una importante área de aplicación para las bases de datos. Las herramientas para analizar grandes cantidades de datos experimentaron un gran crecimiento de uso.

  Muchos vendedores de bases de datos introdujeron productos de bases de datos paralelas en este periodo, así como también comenzaron ofrecer bases de datos relacionales orientadas a objeto.

- **Finales de la década de 1990.** El principal acontecimiento fue el crecimiento explosivo de World Wide Web. Las bases de datos se implantaron mucho más extensivamente que nunca antes. Los sistemas de bases de datos tienen ahora soporte para tasas de transacciones muy altas, así como muy alta fiabilidad y disponibilidad 24×7 (disponibilidad 24 horas al día y 7 días a la semana, que significa que no hay tiempos de inactividad debidos a actividades de mantenimiento planificadas). Los sistemas de bases de datos también tuvieron interfaces Web a los datos.

1.11. RESUMEN

- Un **sistema gestor de bases de datos** (SGBD) consiste en una colección de datos interrelacionados y una colección de programas para acceder a esos datos. Los datos describen una empresa particular.
- El objetivo principal de un SGBD es proporcionar un entorno que sea tanto conveniente como eficiente para las personas que lo usan para la recuperación y almacenamiento de la información.
- Los sistemas de bases de datos se diseñan para almacenar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para el almacenamiento de la información como la provisión de mecanismos para la manipulación de la información. Además, los sistemas de bases de datos deben proporcionar la seguridad de la información almacenada, en caso de caídas del sistema o intentos de accesos sin autorización. Si los datos están compartidos por varios usuarios, el sistema debe evitar posibles resultados anómalos.
- Un propósito principal de un sistema de bases de datos es proporcionar a los usuarios una visión abstracta de los datos. Es decir, el sistema esconde ciertos detalles de cómo los datos se almacenan y mantienen.
- Por debajo de la estructura de la base de datos está el **modelo de datos**: una colección de herramientas conceptuales para describir los datos, las relaciones entre los datos, la semántica de los datos y las restricciones de los datos. El modelo de datos entidad-relación es un modelo de datos ampliamente usado, y proporciona una representación gráfica conveniente para ver los datos, las relaciones y las restricciones. El modelo de datos relacional se usa ampliamente para almacenar datos en las bases de datos. Otros modelos de datos son el modelo de datos orientado a objetos, el relacional orientado a objetos y modelos de datos semiestructurados.
- El diseño general de la base de datos se denomina el **esquema** de la base de datos. Un esquema de base de datos se especifica con un conjunto de definiciones

que se expresan usando un **lenguaje de definición de datos (LDD).**

- Un **lenguaje de manipulación de datos (LMD)** es un lenguaje que permite a los usuarios acceder o manipular los datos. Los LMD no procedimentales, que requieren que un usuario especifique sólo los datos que necesita, se usan ampliamente hoy día.
- Los usuarios de bases de datos se pueden catalogar en varias clases, y cada clase de usuario usa habitualmente diferentes tipos de interfaces de la base de datos.
- Un sistema de bases de datos tiene varios subsistemas:
  - — El subsistema gestor de transacciones es el responsable de asegurar que la base de datos permanezca en un estado consistente (correcto) a pesar de los fallos del sistema. El gestor de transacciones también asegura que las ejecuciones de transacciones concurrentes ocurran sin conflictos.
  - — El subsistema procesador de consultas compila y ejecuta instrucciones LDD y LMD.
  - — El subsistema gestor de almacenamiento es un módulo de programa que proporciona la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicación y las consultas enviadas al sistema.
- Las aplicaciones de bases de datos se dividen normalmente en un parte frontal que se ejecuta en las máquinas cliente y una parte que se ejecuta en el dorsal. En las arquitecturas de dos capas, el frontal se comunica directamente con una base de datos que se ejecuta en el dorsal. En las arquitecturas de tres capas, la parte dorsal se divide asimismo en un servidor de aplicaciones y en un servidor de bases de datos.

TÉRMINOS DE REPASO

- Abstracción de datos.
- Administrador de la base de datos (ADB).
- Aplicaciones de sistemas de bases de datos.
- Concurrencia.
- Diccionario de datos.
- Ejemplar de la base de datos.
- Esquema.
  - — Esquema de la base de datos.
  - — Esquema físico.
  - — Esquema lógico.
- Inconsistencia de datos.
- Independencia física de los datos.
- Lenguajes de bases de datos.
  - — Lenguaje de consultas.
  - — Lenguaje de definición de datos.
- Lenguaje de manipulación de datos.
- Máquinas cliente y servidor.
- Metadatos.
- Modelos de datos.
  - — Modelo de datos orientado a objetos.
  - — Modelo de datos relacional.
  - — Modelo de datos relacional orientado a objetos.
  - — Modelo entidad-relación.
- Programa de aplicación.
- Restricciones de consistencia.
- Sistema de gestión de bases de datos (SGBD).
- Sistemas de archivos.
- Transacciones.
- Vistas de datos.

EJERCICIOS

1.1. ¿Cuáles son las cuatro diferencias principales entre un sistema de procesamiento de archivos y un SGBD?

1.2. En este capítulo se han descrito las diferentes ventajas principales de un sistema gestor de bases de datos. ¿Cuáles son los dos inconvenientes?

1.3. Explíquese la diferencia entre independencia de datos física y lógica.

1.4. Lístense las cinco responsabilidades del sistema gestor de la base de datos. Para cada responsabilidad explíquense los problemas que ocurrirían si no se realizara esa función.

1.5. ¿Cuáles son las cinco funciones principales del administrador de la base de datos?

1.6. Lístense siete lenguajes de programación que sean procedimentales y dos que sean no procedimentales. ¿Qué grupo es más fácil de aprender a usar? Explíquese la respuesta.

1.7. Lístense los seis pasos principales que se deberían dar en la realización de una base de datos para una empresa particular.

1.8. Considérese un *array* de enteros bidimensional de tamaño $n \times m$ que se va a usar en su lenguaje de programación preferido. Usando el *array* como ejemplo, ilústrese la diferencia (a) entre los tres niveles de abstracción y (b) entre esquema y ejemplares.

NOTAS BIBLIOGRÁFICAS

A continuación se listan libros de propósito general, colecciones de artículos de investigación y sitios Web de bases de datos.

Libros de texto que tratan los sistemas de bases de datos incluyen Abiteboul et al. [1995], Date [1995], Elmasri y Navathe [2000], O'Neil y O'Neil [2000], Ramakrishnan y Gehrke [2000] y Ullman [1988]. El tratamiento del procesamiento de transacciones en libros de texto se puede encontrar en Bernstein y Newcomer [1997] y Gray y Reuter [1993].

Varios libros incluyen colecciones de artículos de investigación sobre la gestión de bases de datos. Entre estos están Bancilhon y Buneman [1990], Date [1986], Date [1990], Kim [1995], Zaniolo et al. [1997], y Stonebraker y Hellerstein [1998].

Una revisión de los logros en la gestión de bases de datos y una valoración de los desafíos en la investigación futura aparece en Silberschatz et al. [1990], Silberschatz et al. [1996] y Bernstein et al. [1998]. La página inicial del grupo especial de interés de la ACM en gestión de datos (véase www.acm.org/sigmod) proporciona una gran cantidad de información sobre la investigación en bases de datos. Los sitios Web de los vendedores de bases de datos (véase el apartado Herramientas a continuación) proporciona detalles acerca de sus respectivos productos.

Codd [1970] es el artículo histórico que introdujo el modelo relacional. En Fry y Sibley [1976] y Sibley [1976] se ofrecen discusiones referentes a la evolución de los SGBDs y al desarrollo de la tecnología de bases de datos.

HERRAMIENTAS

Hay un gran número de sistemas de bases de datos comerciales en uso actualmente. Los principales incluyen: DB2 de IBM (www.ibm.com/software/data), Oracle (www.oracle.com), Microsoft SQL Server (www.microsoft.com/sql), Informix (www.informix.com) y Sybase (www.sybase.com). Algunos de estos sistemas están disponibles gratuitamente para uso personal o no comercial, o para desarrollo, pero no para implantación real.

Hay también una serie de sistemas de bases de datos gratuitos/públicos; algunos ampliamente usados incluyen MySQL (www.mysql.com) y PostgresSQL (www.postgresql.org).

Una lista más completa de enlaces a vendedores y otra información se encuentra disponible en la página inicial de este libro en www.research.bell-labs.com/topic/books/db-book.

PARTE

I

MODELOS DE DATOS

Un **modelo de datos** es una colección de herramientas conceptuales para la descripción de datos, relaciones entre datos, semántica de los datos y restricciones de consistencia. En esta parte se estudiarán dos modelos de datos —el modelo entidad-relación y el modelo relacional.

El modelo entidad-relación (E-R) es un modelo de datos de alto nivel. Está basado en una percepción de un mundo real que consiste en una colección de objetos básicos, denominados *entidades*, y de *relaciones* entre estos objetos.

El modelo relaciona es un modelo de menor nivel. Usa una colección de tablas para representar tanto los datos como las relaciones entre los datos. Su simplicidad conceptual ha conducido a su adopción general; actualmente, una vasta mayoría de productos de bases de datos se basan en el modelo relacional. Los diseñadores formulan generalmente el diseño del esquema de la base de datos modelando primero los datos en alto nivel, usando el modelo E-R, y después traduciéndolo al modelo relacional.

Se estudiarán otros modelos de datos más tarde en este libro. El modelo de datos orientado a objetos, por ejemplo, extiende la representación de entidades añadiendo nociones de encapsulación, métodos (funciones) e identidad de objeto. El modelo de datos relacional orientado a objetos combina características del modelo de datos orientado a objetos y del modelo de datos relacional. Los Capítulos 8 y 9 tratan respectivamente estos dos modelos de datos.

CAPÍTULO

2 MODELO ENTIDAD-RELACIÓN

EL modelo de datos **entidad-relación (E-R)** está basado en una percepción del mundo real consistente en objetos básicos llamados *entidades* y de *relaciones* entre estos objetos. Se desarrolló para facilitar el diseño de bases de datos permitiendo la especificación de un *esquema de la empresa* que representa la estructura lógica completa de una base de datos. El modelo de datos E-R es uno de los diferentes modelos de datos semánticos; el aspecto semántico del modelo yace en la representación del significado de los datos. El modelo E-R es extremadamente útil para hacer corresponder los significados e interacciones de las empresas del mundo real con un esquema conceptual. Debido a esta utilidad, muchas herramientas de diseño de bases de datos se basan en los conceptos del modelo E-R.

2.1. CONCEPTOS BÁSICOS

Hay tres nociones básicas que emplea el modelo de datos E-R: conjuntos de entidades, conjuntos de relaciones y atributos.

2.1.1. Conjuntos de entidades

Una **entidad** es una «cosa» u «objeto» en el mundo real que es distinguible de todos los demás objetos. Por ejemplo, cada persona en un desarrollo es una entidad. Una entidad tiene un conjunto de propiedades, y los valores para algún conjunto de propiedades pueden identificar una entidad de forma unívoca. Por ejemplo, el D.N.I. 67.789.901 identifica unívocamente una persona particular en la empresa. Análogamente, se puede pensar en los préstamos bancarios como entidades, y un número de préstamo P-15 en la sucursal de Castellana identifica unívocamente una entidad de préstamo. Una entidad puede ser concreta, como una persona o un libro, o puede ser abstracta, como un préstamo, unas vacaciones o un concepto.

Un **conjunto de entidades** es un conjunto de entidades del mismo tipo que comparten las mismas propiedades, o atributos. El conjunto de todas las personas que son clientes en un banco dado, por ejemplo, se pueden definir como el conjunto de entidades *cliente*. Análogamente, el conjunto de entidades *préstamo* podría representar el conjunto de todos los préstamos concedidos por un banco particular. Las entidades individuales que constituyen un conjunto se llaman la *extensión* del conjunto de entidades. Así, todos los clientes de un banco son la extensión del conjunto de entidades *cliente*.

Los conjuntos de entidades no son necesariamente disjuntos. Por ejemplo, es posible definir el conjunto de entidades de todos los empleados de un banco (*empleado*) y el conjunto de entidades de todos los clientes del banco (*cliente*). Una entidad *persona* puede ser una entidad *empleado*, una entidad *cliente*, ambas cosas, o ninguna.

Una entidad se representa mediante un conjunto de **atributos**. Los atributos describen propiedades que posee cada miembro de un conjunto de entidades. La designación de un atributo para un conjunto de entidades expresa que la base de datos almacena información similar concerniente a cada entidad del conjunto de entidades; sin embargo, cada entidad puede tener su propio valor para cada atributo. Posibles atributos del conjunto de entidades *cliente* son *id-cliente*, *nombre-cliente, calle-cliente* y *ciudad-cliente*. En la vida real, habría más atributos, tales como el número de la calle, el número del portal, la provincia, el código postal, y la comunidad autónoma, pero no se incluyen en el ejemplo simple. Posibles atributos del conjunto de entidades *préstamo* son *número-préstamo* e *importe*.

Cada entidad tiene un **valor** para cada uno de sus atributos. Por ejemplo, una entidad *cliente* en concreto puede tener el valor 32.112.312 para *id-cliente*, el valor Santos para *nombre-cliente*, el valor Mayor para *calle-cliente* y el valor Peguerinos para *ciudad-cliente*.

El atributo *id-cliente* se usa para identificar unívocamente a los clientes, dado que no hay más de un cliente con el mismo nombre, calle y ciudad. En los Estados Unidos, muchas empresas encuentran conveniente usar el número *seguridad-social* de una persona[1] como un

[1] En España se asigna a cada persona del país un número único, denominado número del documento nacional de identidad (D.N.I.) para identificarla unívocamente. Se supone que cada persona tiene un único D.N.I., y no hay dos personas con el mismo D.N.I.

atributo cuyo valor identifica unívocamente a la persona. En general la empresa tendría que crear y asignar un identificador a cada cliente.

Para cada atributo hay un conjunto de valores permitidos, llamados el **dominio**, o el **conjunto de valores**, de ese atributo. El dominio del atributo *nombre-cliente* podría ser el conjunto de todas las cadenas de texto de una cierta longitud. Análogamente, el dominio del atributo *número-préstamo* podría ser el conjunto de todas las cadenas de la forma «P-*n*», donde *n* es un entero positivo.

Una base de datos incluye así una colección de conjuntos de entidades, cada una de las cuales contiene un número de entidades del mismo tipo. En la Figura 2.1 se muestra parte de una base de datos de un banco que consta de dos conjuntos de entidades, *cliente* y *préstamo*.

Formalmente, un atributo de un conjunto de entidades es una función que asigna al conjunto de entidades un dominio. Como un conjunto de entidades puede tener diferentes atributos, cada entidad se puede describir como un conjunto de pares (atributo,valor), un par para cada atributo del conjunto de entidades. Por ejemplo, una entidad concreta *cliente* se puede describir mediante el conjunto {(*id-cliente*, 67.789.901), (*nombre-cliente*, López), (*calle-cliente*, Mayor), (*ciudad-cliente*, Peguerinos)}, queriendo decir que la entidad describe una persona llamada López que tiene D.N.I. número 67.789.901, y reside en la calle Mayor en Peguerinos. Se puede ver, en este punto, que existe una integración del esquema abstracto con el desarrollo real de la empresa que se está modelando. Los valores de los atributos que describen una entidad constituirán una porción significante de los datos almacenados en la base de datos.

Un atributo, como se usa en el modelo E-R, se puede caracterizar por los siguientes tipos de atributo.

- Atributos **simples** y **compuestos**. En los ejemplos considerados hasta ahora, los atributos han sido simples; es decir, no están divididos en subpartes. Los atributos compuestos, en cambio, se pueden dividir en subpartes (es decir, en otros atributos). Por ejemplo, *nombre-cliente* podría estar estructurado como un atributo compuesto consistente en *nombre*, *primer-apellido* y *segundo-apellido*. Usar atributos compuestos en un esquema de diseño es una buena elección si el usuario desea referirse a un atributo completo en algunas ocasiones y, en otras, a algún componente del atributo. Se podrían haber sustituido los atributos del conjunto de entidades *cliente*, *calle-cliente* y *ciudad-cliente*, por el atributo compuesto *dirección-cliente*, con los atributos *calle, ciudad, provincia,* y *código-postal*[2]. Los atributos compuestos ayudan a agrupar los atributos relacionados, haciendo los modelos más claros.

  Nótese también que un atributo compuesto puede aparecer como una jerarquía. Volviendo al ejemplo del atributo compuesto *dirección-cliente*, su componente *calle* puede ser a su vez dividido en *número-calle*, *nombre-calle* y *piso*. Estos ejemplos de atributos compuestos para el conjunto de entidades *cliente* se representa en la Figura 2.2.
- Atributos **monovalorados** y **multivalorados**. Los atributos que se han especificado en los ejemplos tienen todos un valor sólo para una entidad concreta. Por ejemplo, el atributo *número-préstamo* para una entidad préstamo específico, referencia a un único número de préstamo. Tales atributos se llaman **monovalorados**. Puede haber ocasiones en las que un atributo tiene un conjunto de valores para una entidad específica. Considérese un conjunto de entidades *empleado* con el atributo *número-teléfono*. Cualquier empleado particular puede tener cero, uno o más números de teléfono. Este tipo de atributo se llama *multivalorado*. En ellos, se pueden colocar apropiadamente límites inferior y superior en el número de valores en el atributo **multivalorado**. Como otro ejemplo, un atributo *nombre-subordinado* del conjunto de entidades *empleado*

[2] Se asume el formato de *calle-cliente* y *dirección* usado en España, que incluye un código postal numérico llamado «código postal».

Santos	32.112.312	Mayor	Peguerinos
Gómez	01.928.374	Carretas	Cerceda
López	67.789.901	Mayor	Peguerinos
Sotoca	55.555.555	Real	Cádiz
Pérez	24.466.880	Carretas	Cerceda
Valdivieso	96.396.396	Goya	Vigo
Fernández	33.557.799	Jazmín	León

cliente

P-17	1.000
P-23	2.000
P-15	1.500
P-14	1.500
P-19	500
P-11	900
P-16	1.300

préstamo

FIGURA 2.1. Conjunto de entidades *cliente* y *préstamo*.

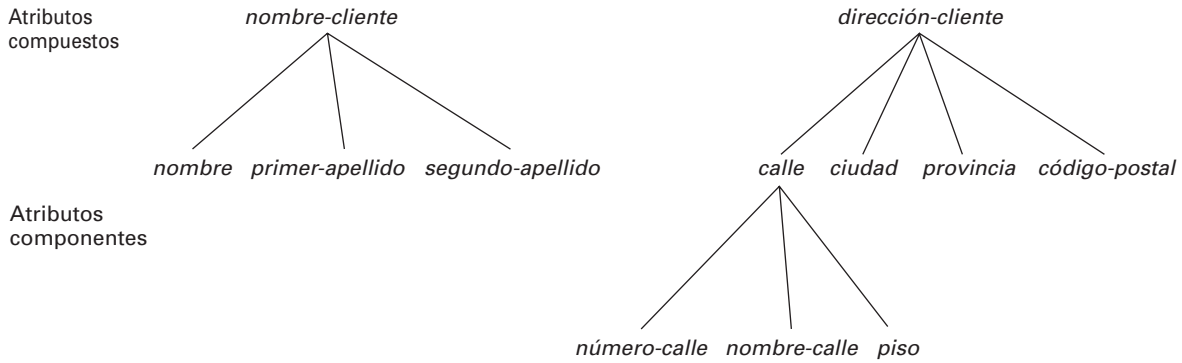

FIGURA 2.2. Atributos compuestos *nombre-cliente* y *dirección-cliente*.

sería multivalorado, ya que un empleado en concreto podría tener cero, uno o más subordinados.

Cuando sea apropiado se pueden establecer límites superior e inferior en el número de valores de un atributo multivalorado. Por ejemplo, un banco puede limitar el número de números de teléfono almacenados para un único cliente a dos. Colocando límites en este caso, se expresa que el atributo *número-teléfono* del conjunto de entidades *cliente* puede tener entre cero y dos valores.

- Atributos **derivados**. El valor para este tipo de atributo se puede derivar de los valores de otros atributos o entidades relacionados. Por ejemplo, sea el conjunto de entidades *cliente* que tiene un atributo *préstamos* que representa cuántos préstamos tiene un cliente en el banco. Ese atributo se puede derivar contando el número de entidades *préstamo* asociadas con ese *cliente*.

  Como otro ejemplo, considérese que el conjunto de entidades *empleado* tiene un atributo *edad*, que indica la edad del cliente. Si el conjunto de entidades *cliente* tiene también un atributo *fecha-de-nacimiento*, se puede calcular *edad* a partir de *fecha-de-nacimiento* y de la fecha actual. Así, *edad* es un atributo derivado. En este caso, *fecha-de-nacimiento* y *antigüedad* pueden serlo, ya que representan el primer día en que el empleado comenzó a trabajar para el banco y el tiempo total que el empleado lleva trabajando para el banco, respectivamente. El valor de *antigüedad* se puede derivar del valor de *fecha-comienzo* y de la fecha actual. En este caso, *fecha-comienzo* se puede conocer como atributo *base* o atributo *almacenado*. El valor de un atributo derivado no se almacena, sino que se calcula cuando sea necesario.

Un atributo toma un valor **nulo** cuando una entidad no tiene un valor para un atributo. El valor *nulo* también puede indicar «no aplicable», es decir, que el valor no existe para la entidad. Por ejemplo, una persona puede no tener segundo nombre de pila. *Nulo* puede también designar que el valor de un atributo es desconocido. Un valor desconocido puede ser, bien *perdido* (el valor existe pero no se tiene esa información) o *desconocido* (no se conoce si el valor existe realmente o no).

Por ejemplo, si el valor *nombre* para un *cliente* particular es *nulo*, se asume que el valor es perdido, ya que cada cliente debe tener un nombre. Un valor *nulo* para el atributo *piso* podría significar que la dirección no incluye un piso (no aplicable), que existe piso pero no se conoce cuál es (perdido), o que no se sabe si el piso forma parte o no de la dirección del cliente (desconocido).

Una base de datos para una empresa bancaria puede incluir diferentes conjuntos de entidades. Por ejemplo, además del mantenimiento de clientes y préstamos, el banco también proporciona cuentas, que se representan mediante el conjunto de entidades *cuenta* con atributos *número-cuenta* y *saldo*. También, si el banco tiene un número de sucursales diferentes, se puede mantener información acerca de todas las sucursales del banco. Cada conjunto de entidades *sucursal* se describe mediante los atributos *nombre-sucursal*, *ciudad-sucursal* y *activo*.

2.1.2. Conjuntos de relaciones

Una **relación** es una asociación entre diferentes entidades. Por ejemplo, se puede definir una relación que asocie al cliente López con el préstamo P-15. Esta relación especifica que López es un cliente con el préstamo número P-15.

Un **conjunto de relaciones** es un conjunto de relaciones del mismo tipo. Formalmente es una relación matemática con $n >= 2$ de conjuntos de entidades (posiblemente no distintos). Si $E_1, E_2, \ldots, E_n$ son conjuntos de entidades, entonces un conjunto de relaciones R es un subconjunto de:

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

donde $(e_1, e_2, \ldots e_n)$ es una relación.

Considérense las dos entidades *cliente* y *préstamo* de la Figura 2.1. Se define el conjunto de relaciones *prestatario* para denotar la asociación entre clientes y préstamos bancarios que los clientes tengan. Esta asociación se describe en la Figura 2.3.

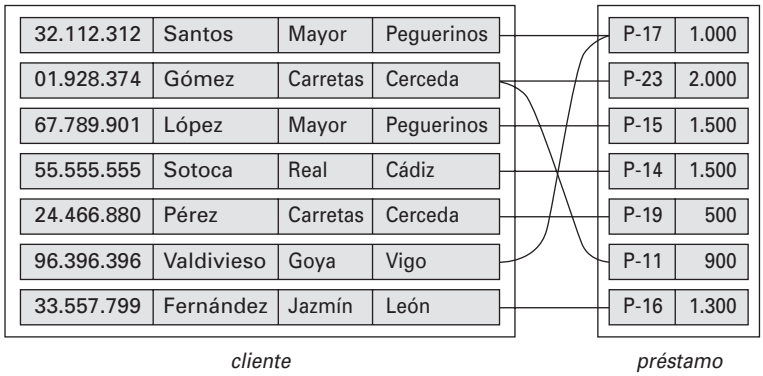

FIGURA 2.3. Conjunto de relaciones *prestatario*.

Como otro ejemplo, considérense los dos conjuntos de entidades *préstamo* y *sucursal*. Se puede definir el conjunto de relaciones *sucursal-préstamo* para denotar la asociación entre un préstamo y la sucursal en que se mantiene ese préstamo.

La asociación entre conjuntos de entidades se conoce como *participación*; es decir, los conjuntos de entidades E_1, E_2,…, E_n **participan** en el conjunto de relaciones R. Un **ejemplar de relación** en un esquema E-R representa que existe una asociación entre las entidades denominadas en la empresa del mundo real que se modela. Como ilustración, el *cliente* individual López, que tiene D.N.I. 67.789.901, y la entidad *préstamo* P-15 participan en un ejemplar de relación de *prestatario*. Este ejemplar de relación representa que, en la empresa del mundo real, la persona llamada López cuyo número de D.N.I. es 67.789.901 ha tomado un préstamo que está numerado como P-15.

La función que desempeña una entidad en una relación se llama **papel** de la entidad.

Debido a que los conjuntos de entidades que participan en un conjunto de relaciones son generalmente distintos, los papeles están implícitos y no se especifican normalmente. Sin embargo, son útiles cuando el significado de una relación necesita aclaración. Tal es el caso cuando los conjuntos de entidades de una relación no son distintos; es decir, el mismo conjunto de entidades participa en una relación más de una vez con diferentes papeles. En este tipo de conjunto de relaciones, que se llama algunas veces conjunto de relaciones **recursivo**, es necesario hacer explícitos los papeles para especificar cómo participa una entidad en un ejemplar de relación. Por ejemplo, considérese una conjunto de entidades *empleado* que almacena información acerca de todos los empleados del banco. Se puede tener un conjunto de relaciones *trabaja-para* que se modela mediante pares ordenados de entidades *empleado*. El primer empleado de un par toma el papel de *trabajador*, mientras el segundo toma el papel de *jefe*. De esta manera, todas las relaciones *trabaja-para* son pares (trabajador, jefe); los pares (jefe, trabajador) están excluidos.

Una relación puede también tener **atributos descriptivos**. Considérese un conjunto de relaciones *impositor* con conjuntos de entidades *cliente* y *cuenta*. Se podría asociar el atributo *fecha-acceso* a esta relación para especificar la fecha más reciente en que un cliente accedió a una cuenta. La relación *impositor* entre las entidades correspondientes al cliente García y la cuenta C-217 se describen mediante {(*fecha-acceso*, 23 mayo 2002)}, lo que significa que la última vez que García accedió a la cuenta C-217 fue el 23 de mayo de 2002.

Como otro ejemplo de atributos descriptivos para relaciones, supóngase que se tienen los conjuntos de entidades *estudiante* y *asignatura* que participan en una relación *matriculado*. Se podría desear almacenar un atributo descriptivo para *créditos* con la relación, para registrar si el estudiante se ha matriculado de la asignatura para obtener créditos o sólo como oyente.

Un ejemplar de relación en un conjunto de relaciones determinado debe ser identificado unívocamente a partir de sus entidades participantes, sin usar los atributos descriptivos. Para comprender este punto supóngase que deseemos modelar todas las fechas en las que un cliente ha accedido a una cuenta. El atributo monovalorado *fecha-acceso* puede almacenar sólo una única fecha de acceso. No se pueden representar varias fechas de acceso por varios ejemplares de relación entre el mismo cliente y cuenta, ya que los ejemplares de relación no estarían identificados unívocamente por las entidades participantes. La forma correcta de manejar este caso es crear un atributo multivalorado *fechas-acceso* que pueda almacenar todas las fechas de acceso.

Sin embargo, puede haber más de un conjunto de relaciones que involucren los mismos conjuntos de entidades. En nuestro ejemplo los conjuntos de entidades *cliente* y *préstamo* participan en el conjunto de relaciones *prestatario*. Además, supóngase que cada préstamo deba tener otro cliente que sirva como avalista para el préstamo. Entonces los conjuntos de entidades *cliente* y *préstamo* pueden participar en otro conjunto de relaciones: *avalista*.

Los conjuntos de relaciones *prestatario* y *sucursal-préstamo* proporcionan un ejemplo de un conjunto de relaciones **binario**, es decir, uno que implica dos conjuntos de entidades. La mayoría de los conjuntos de relaciones en un sistema de bases de datos son binarios.

Ocasionalmente, sin embargo, los conjuntos de relaciones implican más de dos conjuntos de entidades.

Por ejemplo, considérense los conjuntos de entidades *empleado*, *sucursal* y *trabajo*. Ejemplos de las entidades *trabajo* podrían ser director, cajero, auditor y otros. Las entidades trabajo pueden tener los atributos *puesto* y *nivel*. El conjunto de relaciones *trabaja-en* entre *empleado*, *sucursal* y *trabajo* es un ejemplo de una relación ternaria. Una relación ternaria entre Santos, Navacerrada y director indica que Santos actúa de director de la sucursal Navacerrada. Santos también podría actuar como auditor de la sucursal Centro, que estaría representado por otra relación. Podría haber otra relación entre Gómez, Centro y cajero, indicando que Gómez actúa como cajero en la sucursal Centro.

El número de conjuntos de entidades que participan en un conjunto de relaciones es también el **grado** del conjunto de relaciones. Un conjunto de relaciones binario tiene grado 2; un conjunto de relaciones ternario tiene grado 3.

2.2. RESTRICCIONES

Un esquema de desarrollo E-R puede definir ciertas restricciones a las que los contenidos de la base de datos se deben adaptar. En este apartado se examina la correspondencia de cardinalidades y las restricciones de participación, que son dos de los tipos más importantes de restricciones.

2.2.1. Correspondencia de cardinalidades

La **correspondencia de cardinalidades**, o razón de cardinalidad, expresa el número de entidades a las que otra entidad puede estar asociada vía un conjunto de relaciones.

La correspondencia de cardinalidades es la más útil describiendo conjuntos de relaciones binarias, aunque ocasionalmente contribuye a la descripción de conjuntos de relaciones que implican más de dos conjuntos de entidades. Este apartado se centrará en conjuntos de relaciones binarias únicamente.

Para un conjunto de relaciones binarias R entre los conjuntos de entidades A y B, la correspondencia de cardinalidades debe ser una de las siguientes:

- **Uno a uno**. Una entidad en A se asocia con *a lo sumo* una entidad en B, y una entidad en B se asocia con *a lo sumo* una entidad en A (véase la Figura 2.4a).

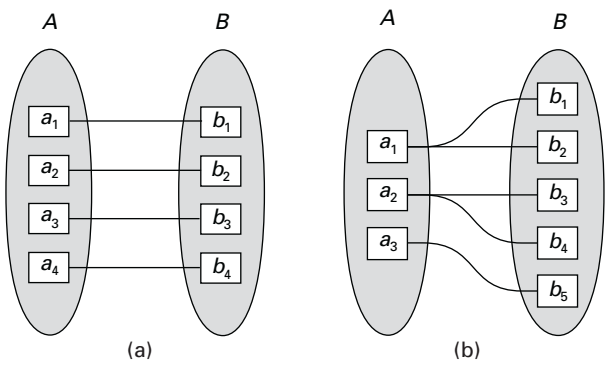

FIGURA 2.4. Correspondencia de cardinalidades. (a) Uno a uno. (b) Uno a varios.

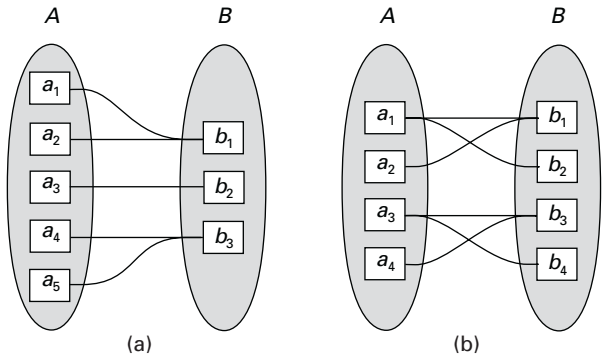

FIGURA 2.5. Correspondencia de cardinalidades. (a) Varios a uno. (b) Varios a varios.

- **Uno a varios**. Una entidad en A se asocia con cualquier número de entidades en B (ninguna o varias). Una entidad en B, sin embargo, se puede asociar con *a lo sumo* una entidad en A (véase la Figura 2.4b).
- **Varios a uno**. Una entidad en A se asocia con *a lo sumo* una entidad en B. Una entidad en B, sin embargo, se puede asociar con cualquier número de entidades (ninguna o varias) en A (véase la Figura 2.5a).
- **Varios a varios**. Una entidad en A se asocia con cualquier número de entidades (ninguna o varias) en B, y una entidad en B se asocia con cualquier número de entidades (ninguna o varias) en A (véase la Figura 2.5b).

La correspondencia de cardinalidades apropiada para un conjunto de relaciones particular depende obviamente de la situación del mundo real que el conjunto de relaciones modela.

Como ilustración considérese el conjunto de relaciones *prestatario*. Si en un banco particular un préstamo puede pertenecer únicamente a un cliente y un cliente puede tener varios préstamos, entonces el conjunto de relaciones de *cliente* a *préstamo* es uno a varios. Si un préstamo puede pertenecer a varios clientes (como préstamos que se toman en conjunto por varios socios de un negocio) el conjunto de relaciones es varios a varios. Este tipo de relación se describe en la Figura 2.3.

2.2.2. Restricciones de participación

La participación de un conjunto de entidades *E* en un conjunto de relaciones *R* se dice que es **total** si cada entidad en *E* participa al menos en una relación en R. Si sólo algunas entidades en *E* participan en relaciones en *R*, la participación del conjunto de entidades *E* en la relación *R* se llama **parcial**. Por ejemplo, se puede esperar que cada entidad *préstamo* esté relacionada con al menos un cliente mediante la relación *prestatario*. Por lo tanto, la participación de *préstamo* en el conjunto de relaciones *prestatario* es total. En cambio, un individuo puede ser cliente de un banco tenga o no tenga un préstamo en el banco. Así, es posible que sólo algunas de las entidades *cliente* estén relacionadas con el conjunto de entidades *préstamo* mediante la relación *prestatario*, y la participación de *cliente* en el conjunto de relaciones *prestatario* es por lo tanto parcial.

2.3. CLAVES

Es necesario tener una forma de especificar cómo las entidades dentro de un conjunto de entidades dado y las relaciones dentro de un conjunto de relaciones dado son distinguibles. Conceptualmente las entidades y relaciones individuales son distintas; desde una perspectiva de bases de datos, sin embargo, la diferencia entre ellas se debe expresar en término de sus atributos.

Por lo tanto, los valores de los atributos de una entidad deben ser tales que permitan *identificar unívocamente* a la entidad. En otras palabras, no se permite que ningún par de entidades tengan exactamente los mismos valores de sus atributos.

Una *clave* permite identificar un conjunto de atributos suficiente para distinguir las entidades entre sí. Las claves también ayudan a identificar unívocamente a las relaciones y así a distinguir las relaciones entre sí.

2.3.1. Conjuntos de entidades

Una **superclave** es un conjunto de uno o más atributos que, tomados colectivamente, permiten identificar de forma única una entidad en el conjunto de entidades. Por ejemplo, el atributo *id-cliente* del conjunto de entidades *cliente* es suficiente para distinguir una entidad *cliente* de las otras. Así, *id-cliente* es una superclave. Análogamente, la combinación de *nombre-cliente* e *id-cliente* es una superclave del conjunto de entidades *cliente*. El atributo *nombre-cliente* de *cliente* no es una superclave, porque varias personas podrían tener el mismo nombre.

El concepto de una superclave no es suficiente para lo que aquí se propone, ya que, como se ha visto, una superclave puede contener atributos innecesarios. Si *K* es una superclave, entonces también lo es cualquier superconjunto de *K*. A menudo interesan las superclaves tales que los subconjuntos propios de ellas no son superclave. Tales superclaves mínimas se llaman **claves candidatas**.

Es posible que conjuntos distintos de atributos pudieran servir como clave candidata. Supóngase que una combinación de *nombre-cliente* y *calle-cliente* es suficiente para distinguir entre los miembros del conjunto de entidades *cliente*. Entonces, los conjuntos {*id-cliente*} y {*nombre-cliente*, *calle-cliente*} son claves candidatas. Aunque los atributos *id-cliente* y *nombre-cliente* juntos puedan distinguir entidades *cliente*, su combinación no forma una clave candidata, ya que el atributo *id-cliente* por sí solo es una clave candidata.

Se usará el término **clave primaria** para denotar una clave candidata que es elegida por el diseñador de la base de datos como elemento principal para identificar las entidades dentro de un conjunto de entidades. Una clave (primaria, candidata y superclave) es una propiedad del conjunto de entidades, más que de las entidades individuales. Cualesquiera dos entidades individuales en el conjunto no pueden tener el mismo valor en sus atributos clave al mismo tiempo. La designación de una clave representa una restricción en el desarrollo del mundo real que se modela.

Las claves candidatas se deben designar con cuidado. Como se puede comprender, el nombre de una persona es obviamente insuficiente, ya que hay mucha gente con el mismo nombre. En España, el D.N.I. puede ser una clave candidata. Como los no residentes en España normalmente no tienen D.N.I., las empresas internacionales pueden generar sus propios identificadores únicos. Una alternativa es usar alguna combinación única de otros atributos como clave.

La clave primaria se debería elegir de manera que sus atributos nunca, o muy raramente, cambien. Por ejemplo, el campo dirección de una persona no debería formar parte de una clave primaria, porque probablemente cambiará. Los números de D.N.I., por otra parte, es seguro que no cambiarán. Los identificadores únicos generados por empresas generalmente no cambian, excepto si se fusionan dos empresas; en tal caso el mismo identificador puede haber sido emitido por ambas empresas y es necesario la reasignación de identificadores para asegurarse de que sean únicos.

2.3.2. Conjuntos de relaciones

La clave primaria de un conjunto de entidades permite distinguir entre las diferentes entidades del conjunto. Se necesita un mecanismo similar para distinguir entre las diferentes relaciones de un conjunto de relaciones.

Sea *R* un conjunto de relaciones que involucra los conjuntos de entidades E_1, E_2,…, E_n. Sea *clave-prima-*

ria(E_i) el conjunto de atributos que forma la clave primaria para el conjunto de entidades E_i.

Asúmase por el momento que los nombres de los atributos de todas las claves primarias son únicos y que cada conjunto de entidades participa sólo una vez en la relación. La composición de la clave primaria para un conjunto de relaciones depende de la estructura de los atributos asociados al conjunto de relaciones *R*.

Si el conjunto de relaciones *R* no tiene atributos asociados, entonces el conjunto de atributos:

$$\textit{clave-primaria}(E_1) \cup \textit{clave-primaria}(E_2) \cup \ldots \cup \textit{clave-primaria}(E_n)$$

describe una relación individual en el conjunto *R*.

Si el conjunto de relaciones *R* tiene atributos $a_1, a_2, \ldots, a_m$ asociados a él, entonces el conjunto de atributos

$$\textit{clave-primaria}(E_1) \cup \textit{clave-primaria}(E_2) \cup \ldots \cup \textit{clave-primaria}(E_n) \cup \{a_1, a_2, \ldots, a_m\}$$

describe una relación individual en el conjunto *R*.

En ambos casos, el conjunto de atributos

$$\textit{clave-primaria}(E_1) \cup \textit{clave-primaria}(E_2) \cup \ldots \cup \textit{clave-primaria}(E_n)$$

forma una superclave para el conjunto de relaciones.

En el caso de que los nombres de atributos de las claves primarias no sean únicos en todos los conjuntos de entidades, los atributos se renombran para distinguirlos; el nombre del conjunto de entidades combinado con el atributo formaría un nombre único. En el caso de que un conjunto de entidades participe más de una vez en un conjunto de relaciones (como en la relación *trabaja-para* del Apartado 2.1.2) el nombre del papel se usa en lugar del nombre del conjunto de entidades para formar un nombre único de atributo.

La estructura de la clave primaria para el conjunto de relaciones depende de la correspondencia de cardinalidades asociada al conjunto de relaciones. Como ilustración, considérese el conjunto de entidades *cliente* y *cuenta*, y un conjunto de relaciones *impositor*, con el atributo *fecha-acceso* del Apartado 2.1.2. Supóngase que el conjunto de relaciones es varios a varios. Entonces la clave primaria de *impositor* consiste en la unión de las claves primarias de *cliente* y *cuenta*. Sin embargo, si un cliente puede tener sólo una cuenta —es decir, si la relación *impositor* es varios a uno de *cliente* a *cuenta*— entonces la clave primaria de *impositor* es simplemente la clave primaria de *cliente*. Análogamente, si la relación es varios a uno de *cuenta* a *cliente* —es decir, cada cuenta pertenece a lo sumo a un cliente— entonces la clave primaria de *impositor* es simplemente la clave primaria de *cuenta*. Para relaciones uno a uno se puede usar cualquier clave primaria.

Para las relaciones no binarias, si no hay restricciones de cardinalidad, entonces la superclave formada como se describió anteriormente en este apartado es la única clave candidata, y se elige como clave primaria. La elección de la clave primaria es más complicada si aparecen restricciones de cardinalidad. Ya que no se ha discutido cómo especificar restricciones de cardinalidad en relaciones no binarias, no se discutirá este aspecto en este capítulo. Se considerará este aspecto con más detalle en el apartado 7.3.

2.4. CUESTIONES DE DISEÑO

Las nociones de conjunto de entidades y conjunto de relaciones no son precisas, y es posible definir un conjunto de entidades y las relaciones entre ellas de diferentes formas. En este apartado se examinan cuestiones básicas de diseño de un esquema de bases de datos E-R. El proceso de diseño se trata con más detalle en el Apartado 2.7.4.

2.4.1. Uso de conjuntos de entidades o atributos

Considérese el conjunto de entidades *empleado* con los atributos *nombre-empleado* y *número-teléfono*. Se puede argumentar fácilmente que un *teléfono* es una entidad por sí misma con atributos *número-teléfono* y *ubicación* (la oficina donde está ubicado el teléfono). Si se toma este punto de vista, el conjunto de entidades *empleado* debe ser redefinido como sigue:

- El conjunto de entidades *empleado* con el atributo *nombre-empleado*
- El conjunto de entidades *teléfono* con atributos *número-teléfono* y *ubicación*
- La relación *empleado-teléfono*, que denota la asociación entre empleados y los teléfonos que tienen.

¿Cuál es, entonces, la diferencia principal entre esas dos definiciones de un empleado? Al tratar un teléfono como un atributo *número-teléfono* implica que cada empleado tiene precisamente un número de teléfono. Al tratar un teléfono como una entidad *teléfono* permite que los empleados puedan tener varios números de teléfono (incluido ninguno) asociados a ellos. Sin embargo, se podría definir fácilmente *número-teléfono* como un atributo multivalorado para permitir varios teléfonos por empleado.

La diferencia principal es que al tratar un teléfono como una entidad se modela mejor una situación en la que se puede querer almacenar información extra sobre un teléfono, como su ubicación, su tipo (móvil, videoteléfono o fijo) o quiénes comparten un teléfono. Así, al tratar un teléfono como una entidad es más general que tratarlo como un atributo y es apropiado cuando la generalidad pueda ser de utilidad.

En cambio, no sería adecuado tratar el atributo *nombre-empleado* como una entidad; es difícil argumentar que *nombre-empleado* sea una entidad por sí mismo (a diferencia del teléfono). Así, es apropiado tener *nombre-empleado* como un atributo del conjunto de entidades *empleado*.

Por tanto, aparecen dos cuestiones naturales: ¿qué constituye un atributo? y ¿qué constituye un conjunto de entidades? Por desgracia no hay respuestas simples. Las distinciones dependen principalmente de la estructura de la empresa del mundo real que se esté modelando y de la semántica asociada con el atributo en cuestión.

Un error común es usar la clave primaria de un conjunto de entidades como un atributo de otro conjunto de entidades, en lugar de usar una relación. Por ejemplo, es incorrecto modelar *id-cliente* como un atributo de *préstamo* incluso si cada préstamo tiene sólo un cliente. La relación *prestatario* es la forma correcta de representar la conexión entre préstamos y clientes, ya que hace su conexión explícita en lugar de implícita mediante un atributo.

Otro error relacionado que se comete es designar a los atributos de la clave primaria de los conjuntos de entidades relacionados como atributos del conjunto de relaciones. Esto no se debería hacer, ya que los atributos de la clave primaria son ya implícitos en la relación.

2.4.2. Uso de conjuntos de entidades o conjuntos de relaciones

No siempre está claro si es mejor expresar un objeto mediante un conjunto de entidades o mediante un conjunto de relaciones. En el Apartado 2.1.1 se asumió que un préstamo se modelaba como una entidad. Una alternativa es modelar un préstamo no como una entidad, sino como una relación entre clientes y sucursales, con *número-préstamo* e *importe* como atributos descriptivos. Cada préstamo se representa mediante una relación entre un cliente y una sucursal.

Si cada préstamo está asociado exactamente con un cliente y con una sucursal, se puede encontrar satisfactorio el diseño en el que un préstamo se representa como una relación. Sin embargo, con este diseño no se puede representar convenientemente una situación en que varios clientes comparten un préstamo. Habría que definir una relación separada para cada prestatario de ese préstamo común. Entonces habría que replicar los valores para los atributos descriptivos *número-préstamo* e *importe* en cada una de estas relaciones. Cada una de estas relaciones debe, por supuesto, tener el mismo valor para los atributos descriptivos *número-préstamo* e *importe*.

Surgen dos problemas como resultado de esta réplica: 1) los datos se almacenan varias veces, desperdiciando espacio de almacenamiento; y 2) las actualizaciones dejan potencialmente los datos en un estado inconsistente, en el que los valores difieren en dos relaciones para atributos que se supone tienen el mismo valor. El asunto de cómo evitar esta réplica se trata formalmente mediante la *teoría de la normalización*, discutida en el Capítulo 7.

El problema de la réplica de los atributos *número-préstamo* e *importe* no aparece en el diseño original del Apartado 2.1.1. porque *préstamo* es un conjunto de entidades.

Una posible guía para determinar si usar un conjunto de entidades o un conjunto de relaciones es designar un conjunto de relaciones para describir una acción que ocurre entre entidades. Este enfoque puede también ser útil para decidir si ciertos atributos se pueden expresar más apropiadamente como relaciones.

2.4.3. Conjuntos de relaciones binarias o *n*-arias

Las relaciones en las bases de datos son generalmente binarias. Algunas relaciones que parecen no ser binarias podrían ser representadas mejor con varias relaciones binarias. Por ejemplo, uno podría crear una relación ternaria *padres*, que relaciona un hijo con su padre y su madre. Sin embargo, tal relación se podría representar por dos relaciones binarias *padre* y *madre*, relacionando un hijo con su padre y su madre por separado. Al usar las dos relaciones *padre* y *madre* se permite registrar la madre de un niño incluso si no se conoce la identidad del padre; en la relación ternaria *padres* se necesitaría usar un valor nulo. En este caso es preferible usar conjuntos de relaciones binarias.

De hecho, siempre es posible reemplazar un conjunto de relaciones no binarias (n-aria, para $n > 2$) por un número de diferentes conjuntos de relaciones binarias. Por simplicidad, considérese el conjunto de relaciones abstracto R, ternario ($n = 3$), y los conjuntos de entidades A, B, y C. Se sustituye el conjunto de relaciones R por un conjunto de entidades E y se crean tres conjuntos de relaciones:

- R_A, relacionando E y A
- R_B, relacionando E y B
- R_C, relacionando E y C

Si el conjunto de relaciones R tiene atributos, éstos se asignan al conjunto de entidades E; por otra parte se crea un atributo de identificación especial para E (debido a que cada conjunto de entidades debe tener al menos un atributo para distinguir los miembros del conjunto).

Para cada relación (a_i,b_i,c_i) del conjunto de relaciones R, se crea una nueva entidad e_i en el conjunto de entidades E. Entonces, en cada uno de los tres nuevos conjuntos de relaciones, se inserta un nuevo miembro como sigue:

- (e_i,a_i) en R_A
- (e_i,b_i) en R_B
- (e_i,c_i) en R_C

Se puede generalizar este proceso de una forma semejante a conjuntos de relaciones n-arias. Así, conceptualmente, se puede restringir el modelo E-R para incluir sólo conjuntos de relaciones binarias. Sin embargo, esta restricción no siempre es deseable.

- Un atributo de identificación puede haber sido creado para el conjunto de entidades para representar el conjunto de relaciones. Este atributo, con los conjuntos de relaciones extra necesarios, incrementa la complejidad del diseño y (como se verá en el Apartado 2.9) los requisitos de almacenamiento.
- Un conjunto de relaciones n-arias muestra más claramente que varias entidades participan en una relación simple.
- Podría no haber una forma de traducir restricciones en la relación ternaria en restricciones sobre relaciones binarias. Por ejemplo, considérese una restricción que dice que R es varios a uno de A, B a C; es decir, cada par de entidades de A y B se asocia con a lo sumo una entidad C. Esta restricción no se puede expresar usando restricciones de cardinalidad sobre los conjuntos de relaciones R_A, R_B y R_C.

Considérese el conjunto de relaciones *trabaja-en* del Apartado 2.1.2 que relaciona *empleado*, *sucursal* y *trabajo*. No se puede dividir directamente *trabaja-en* en relaciones binarias entre *empleado* y *sucursal* y entre *empleado* y *trabajo*. Si se hiciese habría que registrar que Santos es director y auditor y que Santos trabaja en Navacerrada y Centro; sin embargo, no se podría registrar que Santos es director de Navacerrada y auditor de Centro, pero que no es auditor de Navacerrada y director de Centro.

El conjunto de relaciones *trabaja-en* se puede dividir en relaciones binarias creando nuevos conjuntos de entidades como se describió anteriormente. Sin embargo, no sería muy natural.

2.4.4. Ubicación de los atributos de las relaciones

La razón de cardinalidad de una relación puede afectar a la situación de los atributos de la relación. Los atributos de los conjuntos de relaciones uno a uno o uno a varios pueden estar asociados con uno de los conjuntos de entidades participantes, en lugar de con el conjunto de relaciones. Por ejemplo, especificamos que *impositor* es un conjunto de relaciones uno a varios tal que un cliente puede tener varias cuentas, pero cada cuenta está asociada únicamente con un cliente. En este caso, el atributo *fecha-acceso*, que especifica cuándo accedió por última vez el cliente a la cuenta, podría estar asociado con el conjunto de entidades *cuenta*, como se describe en la Figura 2.6; para mantener la simplicidad de la figura sólo se muestran algunos de los atributos de los dos conjuntos de entidades. Como cada entidad *cuenta* participa en una relación con a lo sumo un ejemplar de *cliente*, hacer esta designación de atributos tendría el mismo significado que si se colocase *fecha-acceso* en el conjunto de relaciones *impositor*. Los atributos de un conjunto de relaciones uno a varios se pueden colocar sólo en el conjunto de entidades de la parte «varios» de la relación. Por otra parte, para los conjuntos de entidades uno a uno, los atributos de la relación se pueden asociar con cualquiera de las entidades participantes.

La decisión de diseño de dónde colocar los atributos descriptivos en tales casos —como un atributo de la relación o de la entidad— podría reflejar las características de la empresa que se modela. El diseñador puede elegir mantener *fecha-acceso* como un atributo de *impositor* para expresar explícitamente que ocurre un acceso en el punto de interacción entre los conjuntos de entidades *cliente* y *cuenta*.

La elección de la colocación del atributo es más clara para los conjuntos de relaciones varios a varios. Volviendo al ejemplo, especificamos el caso quizá más realista de *impositor* que es un conjunto de relaciones varios a varios, expresando que un cliente puede tener una o más cuentas, y que una cuenta puede ser mantenida por uno o más clientes. Si se expresa la fecha en que un cliente específico accedió por última vez a una cuenta específica, *fecha-acceso* debe ser un atributo del conjunto de relaciones *impositor*, en lugar de una de las entidades participantes. Si *fecha-acceso* fuese un atributo de *cuenta*, por ejemplo, no se podría determinar

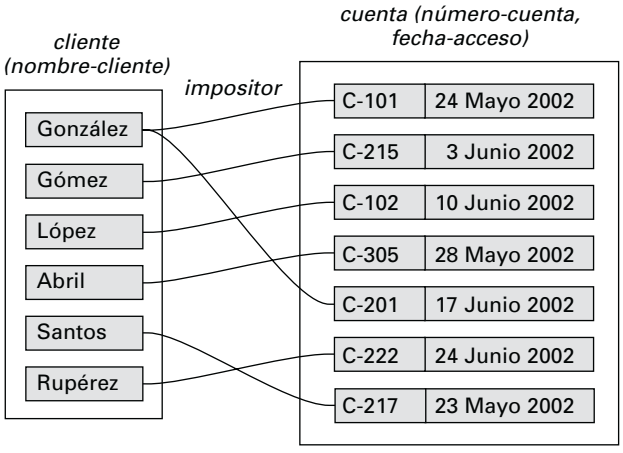

FIGURA 2.6. *Fecha-acceso* como atributo del conjunto de entidades *cuenta*.

qué cliente hizo el acceso más reciente a una cuenta conjunta. Cuando un atributo se determina mediante la combinación de los conjuntos de entidades participantes, en lugar de por cada entidad por separado, ese atributo debe estar asociado con el conjunto de relaciones varios a varios. La colocación de *fecha-acceso* como un atributo de la relación se describe en la Figura 2.7; de nuevo, para mantener la simplicidad de la figura, sólo se muestran algunos de los atributos de los dos conjuntos de entidades.

2.5. DIAGRAMA ENTIDAD-RELACIÓN

Como se vio brevemente en el Apartado 1.4, la estructura lógica general de una base de datos se puede expresar gráficamente mediante un **diagrama E-R**. Los diagramas son simples y claros, cualidades que pueden ser responsables del amplio uso del modelo E-R. Tal diagrama consta de los siguientes componentes principales:

- **Rectángulos**, que representan conjuntos de entidades.
- **Elipses**, que representan atributos.
- **Rombos**, que representan relaciones.
- **Líneas**, que unen atributos a conjuntos de entidades y conjuntos de entidades a conjuntos de relaciones.
- **Elipses dobles**, que representan atributos multivalorados.
- **Elipses discontinuas**, que denotan atributos derivados.
- **Líneas dobles**, que indican participación total de una entidad en un conjunto de relaciones.
- **Rectángulos dobles**, que representan conjuntos de entidades débiles (se describirán posteriormente en el Apartado 2.6).

Considérese el diagrama entidad-relación de la Figura 2.8, que consta de dos conjuntos de entidades, *cliente* y *préstamo*, relacionadas a través de un conjunto de relaciones binarias *prestatario*. Los atributos asociados con *cliente* son *id-cliente*, *nombre-cliente*, *calle-cliente*, y *ciudad-cliente*. Los atributos asociados con *préstamo* son *número-préstamo* e *importe*. Como se muestra en la Figura 2.8, los atributos de un conjunto de entidades que son miembros de la clave primaria están subrayados.

El conjunto de relaciones *prestatario* puede ser varios a varios, uno a varios, varios a uno o uno a uno. Para distinguir entre estos tipos, se dibuja o una línea dirigida (→) o una línea no dirigida (—) entre el conjunto de relaciones y el conjunto de entidades en cuestión.

- Una línea dirigida desde el conjunto de relaciones *prestatario* al conjunto de entidades *préstamo* espe-

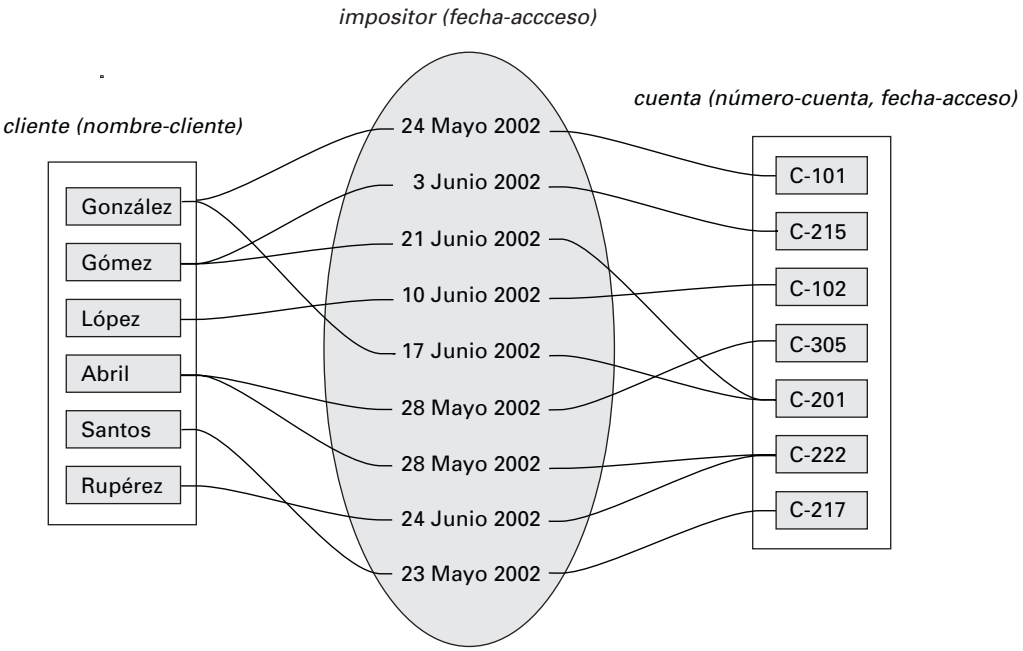

FIGURA 2.7. *Fecha-acceso* como atributo del conjunto de relaciones *impositor*.

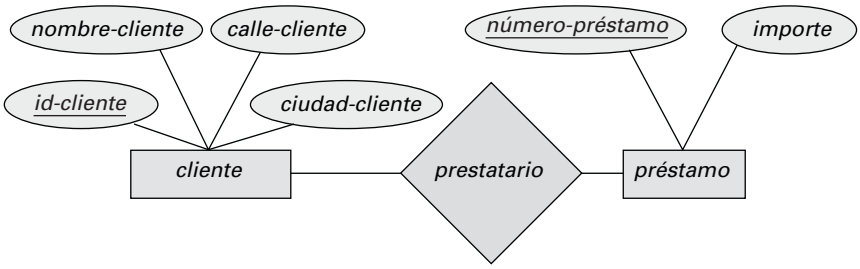

FIGURA 2.8. Diagrama E-R correspondiente a clientes y préstamos.

cifica que *prestatario* es un conjunto de relaciones uno a uno, o bien varios a uno, desde *cliente* a *préstamo*; *prestatario* no puede ser un conjunto de relaciones varios a varios ni uno a varios, desde *cliente* a *préstamo*.

- Una línea no dirigida desde el conjunto de relaciones *prestatario* al conjunto de relaciones *préstamo* especifica que *prestatario* es o bien un conjunto de relaciones varios a varios, o bien uno a varios, desde *cliente* a *préstamo*.

Volviendo al diagrama E-R de la Figura 2.8, se ve que el conjunto de relaciones *prestatario* es varios a varios. Si el conjunto de relaciones *prestatario* fuera uno a varios, desde *cliente* a *préstamo*, entonces la línea desde *prestatario* a *cliente* sería dirigida, con una flecha apuntando al conjunto de entidades *cliente* (Figura 2.9a). Análogamente, si el conjunto de relaciones *pres-*

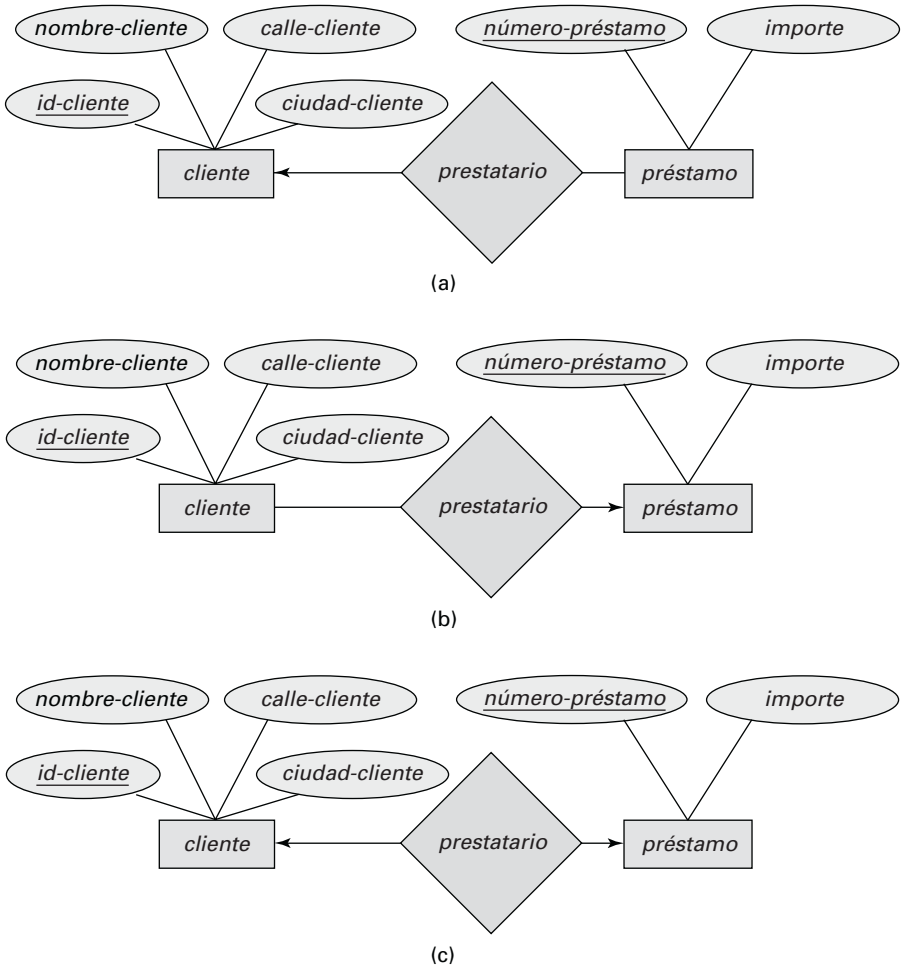

FIGURA 2.9. Relaciones. (a) Uno a varios. (b) Varios a uno. (c) Uno a uno.

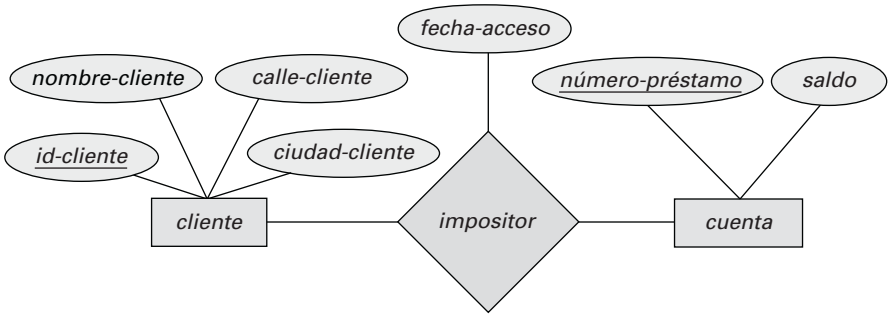

FIGURA 2.10. Diagrama E-R con un atributo unido a un conjunto de relaciones.

tatario fuera varios a uno desde *cliente* a *préstamo*, entonces la línea desde *prestatario* a *préstamo* tendría una flecha apuntando al conjunto de entidades *préstamo* (Figura 2.9b). Finalmente, si el conjunto de relaciones *prestatario* fuera uno a uno, entonces ambas líneas desde *prestatario* tendrían flechas: una apuntando al conjunto de entidades *préstamo* y otra apuntando al conjunto de entidades *cliente* (Figura 2.9c).

Si un conjunto de relaciones tiene también algunos atributos asociados a él, entonces se unen esos atributos a ese conjunto de relaciones. Por ejemplo, en la Figura 2.10, se tiene el atributo descriptivo *fecha-acceso* unido al conjunto de relaciones *impositor* para especificar la fecha más reciente en la que un cliente accedió a esa cuenta.

La Figura 2.11 muestra cómo se pueden representar atributos compuestos en la notación E-R. Aquí, el atributo compuesto *nombre*, con atributos componentes *nombre-pila*, *primer-apellido* y *segundo-apellido* reemplaza al atributo simple *nombre-cliente* de *cliente*. También se muestra el atributo compuesto *dirección*, cuyos atributos componentes son *calle*, *ciudad*, *provincia* y *código-postal*, que reemplaza a los atributos *calle-cliente* y *ciudad-cliente* de *cliente*. El atributo *calle* es por sí mismo un atributo compuesto cuyos atributos componentes son *número-calle*, *nombre-calle* y *número-piso*.

La Figura 2.11 también muestra un atributo multivalorado, *número-teléfono*, indicado por una elipse doble, y un atributo derivado *edad*, indicado por una elipse discontinua.

En los diagramas E-R se indican papeles mediante etiquetas en las líneas que unen rombos con rectángulos. En la Figura 2.12 se muestran los indicadores de papeles *director* y *trabajador* entre el conjunto de entidades *empleado* y el conjunto de relaciones *trabaja-para*.

Los conjuntos de relaciones no binarias se pueden especificar fácilmente en un diagrama E-R. La Figura 2.13 consta de tres conjuntos de entidades *cliente*, *trabajo* y *sucursal*, relacionados a través del conjunto de relaciones *trabaja-en*.

Se pueden especificar algunos tipos de relaciones varios a uno en el caso de conjuntos de relaciones no binarias. Supóngase un empleado que tenga a lo sumo un trabajo en cada sucursal (por ejemplo, Santos no puede ser director y auditor en la misma sucursal). Esta restricción se puede especificar con una flecha apuntando a *trabajo* en el borde de *trabaja-en*.

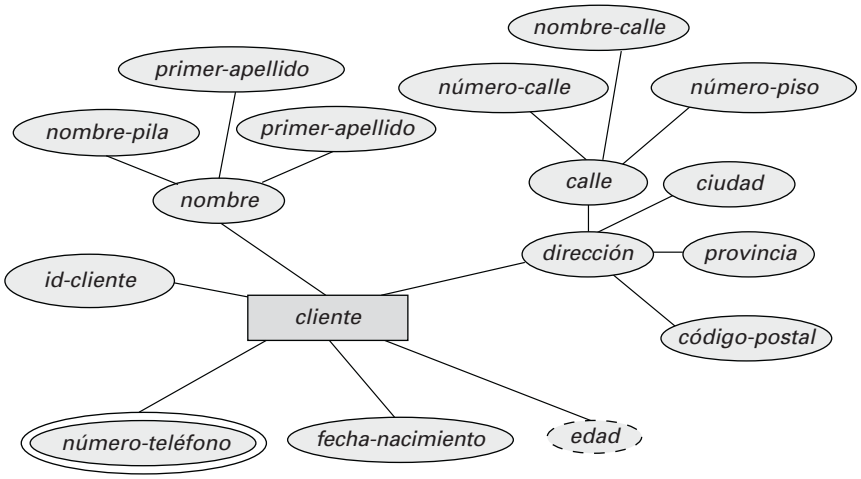

FIGURA 2.11. Diagrama E-R con atributos compuestos, multivalorados y derivados.

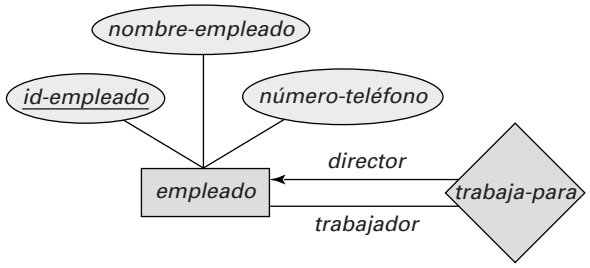

FIGURA 2.12. Diagrama E-R con indicadores de papeles.

Se permite a lo sumo una flecha desde un conjunto de relaciones, ya que un diagrama E-R con dos o más flechas salientes de un conjunto de relaciones no binarias se puede interpretar de dos formas. Supónganse que hay un conjunto de relaciones R entre conjuntos de entidades $A_1, A_2, \ldots, A_n$, y las únicas flechas están en los bordes de los conjuntos de entidades $A_{i+1}, A_{i+2}, \ldots, A_n$. Entonces, las dos posibles interpretaciones son:

1. Una combinación particular de entidades de $A_1, A_2, \ldots, A_i$ se puede asociar con a lo sumo una combinación de entidades de $A_{i+1}, A_{i+2}, \ldots, A_n$. Así, la clave primaria de la relación R se puede construir por la unión de las claves primarias de $A_1, A_2, \ldots, A_i$.
2. Para cada conjunto de entidades A_k, $i < k \leq = n$, cada combinación de las entidades de los otros conjuntos de entidades se pueden asociar con a lo sumo una entidad de A_k. Cada conjunto $\{A_1, A_2, \ldots, A_{k-1}, A_{k+1}, A_{i+2}, \ldots, A_n\}$, para $i < k \leq = n$, forma entonces una clave candidata.

Cada una de estas interpretaciones se han usado en diferentes libros y sistemas. Para evitar confusión se permite sólo una flecha que salga de un conjunto de relaciones, y así las representaciones son equivalentes. En el Capítulo 7 (Apartado 7.3) se estudia la noción de *dependencias funcionales*, que permiten especificar cualquiera de estas dos interpretaciones sin ambigüedad.

En el diagrama E-R se usan las líneas dobles para indicar que la participación de un conjunto de entidades en un conjunto de relaciones es total; es decir, cada entidad en el conjunto de entidades aparece al menos en una relación en ese conjunto de relaciones. Por ejemplo, considérese la relación *prestamista* entre clientes y préstamos. Una línea doble de *préstamo* a *prestamista*, como en la Figura 2.14, indica que cada préstamo debe tener al menos un cliente asociado.

Los diagramas E-R también proporcionan una forma de indicar restricciones más complejas sobre el número de veces en que cada entidad participa en las relaciones de un conjunto de relaciones. Un segmento entre un conjunto de entidades y un conjunto de relaciones binarias puede tener una cardinalidad mínima y máxima, mostrada de la forma *mín..máx*, donde *mín* es la mínima cardinalidad y *máx* es la máxima. Un valor mínimo de 1 indica una participación total del conjunto de entidades en el conjunto de relaciones. Un valor máximo de 1 indica que la entidad participa de a lo sumo una relación, mientras que un valor máximo de * indica que no hay límite. Nótese que una etiqueta 1..* en un segmento es equivalente a una línea doble.

Por ejemplo, considérese la Figura 2.15. El segmento entre *préstamo* y *prestamista* tiene una restricción de cardinalidad de 1..1, significando que la cardinalidad mínima y máxima son ambas 1. Es decir, cada préstamo debe tener exactamente un cliente asociado. El límite 0..* en el segmento de *cliente* a *prestamista* indica que un cliente puede tener ninguno o varios préstamos. Así, la relación *prestamista* es uno a varios de *cliente* a *préstamo*, y además la participación de *préstamo* en *prestamista* es total.

Es fácil malinterpretar 0..* en el segmento entre *cliente* y *prestamista*, y pensar que la relación *prestamista* es de varios a uno de *cliente* a *préstamo* —esto es exactamente lo contrario a la interpretación correcta.

Si ambos segmentos de una relación binaria tienen un valor máximo de 1, la relación es uno a uno. Si se hubiese especificado un límite de cardinalidad de 1..* en el segmento entre *cliente* y *prestamista*, se estaría diciendo que cada cliente debe tener al menos un préstamo.

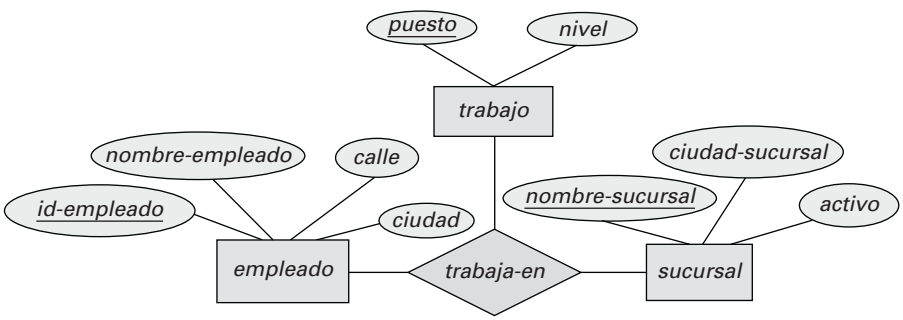

FIGURA 2.13. Diagrama E-R con una relación ternaria.

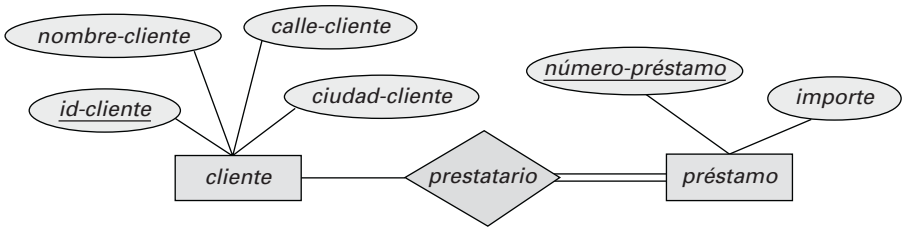

FIGURA 2.14. Participación total de un conjunto de entidades en un conjunto de relaciones.

2.6. CONJUNTOS DE ENTIDADES DÉBILES

Un conjunto de entidades puede no tener suficientes atributos para formar una clave primaria. Tal conjunto de entidades se denomina **conjunto de entidades débiles**. Un conjunto de entidades que tiene una clave primaria se denomina **conjunto de entidades fuertes**.

Como ilustración, considérese el conjunto de entidades *pago*, que tiene los tres atributos: *número-pago, fecha-pago* e *importe-pago*. Los números de pago son generalmente números secuenciales, empezando por 1, generados por separado por cada préstamo. Así, aunque cada entidad *pago* es distinta, los pagos para diferentes préstamos pueden compartir el mismo número de pago. Así, este conjunto de entidades no tiene una clave primaria; es un conjunto de entidades débiles.

Para que un conjunto de entidades débiles tenga sentido, debe estar asociada con otro conjunto de entidades, denominado el **conjunto de entidades identificadoras** o **propietarias**. Cada entidad débil debe estar asociada con una entidad identificadora; es decir, se dice que el conjunto de entidades débiles **depende existencialmente** del conjunto de entidades identificadoras. Se dice que el conjunto de entidades identificadoras es **propietaria** del conjunto de entidades débiles que identifica. La relación que asocia el conjunto de entidades débiles con el conjunto de entidades identificadoras se denomina **relación identificadora**. La relación identificadora es varios a uno del conjunto de entidades débiles al conjunto de entidades identificadoras y la participación del conjunto de entidades débiles en la relación es total.

En nuestro ejemplo, el conjunto de entidades identificador para *pago* es *préstamo*, y la relación *préstamo-pago* que asocia las entidades *pago* con sus correspondientes entidades *préstamo* es la relación identificadora.

Aunque un conjunto de entidades débiles no tiene clave primaria, no obstante se necesita conocer un medio para distinguir todas aquellas entidades del conjunto de entidades que dependen de una entidad fuerte particular. El **discriminante** de un conjunto de entidades débiles es un conjunto de atributos que permite que esta distinción se haga. Por ejemplo, el discriminante del conjunto de entidades débiles *pago* es el atributo *número-pago*, ya que, para cada préstamo, un número de pago identifica de forma única cada pago para ese préstamo. El discriminante de un conjunto de entidades débiles se denomina la *clave parcial* del conjunto de entidades.

La clave primaria de un conjunto de entidades débiles se forma con la clave primaria del conjunto de entidades identificadoras, más el discriminante del conjunto de entidades débiles. En el caso del conjunto de entidades *pago*, su clave primaria es {*número-préstamo*, *número-pago*}, donde *número-préstamo* es la clave primaria del conjunto de entidades identificadoras, es decir, *préstamo*, y *número-pago* distingue las entidades *pago* dentro del mismo préstamo.

El conjunto de entidades identificadoras no debería tener atributos descriptivos, ya que cualquier atributo requerido puede estar asociado con el conjunto de enti-

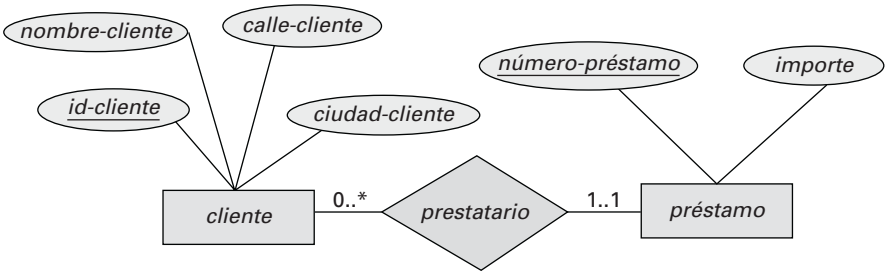

FIGURA 2.15. Límites de cardinalidad en conjuntos de relaciones.

dades débiles (véase la discusión de trasladar los atributos del conjunto de relaciones a los conjuntos de entidades participantes en el Apartado 2.2.1).

Un conjunto de entidades débiles puede participar en relaciones distintas de relaciones identificadoras. Por ejemplo, la entidad *pago* podría participar en una relación con el conjunto de entidades con el conjunto de entidades *cuenta*, identificando la cuenta desde la que se realizó el pago. Un conjunto de entidades débiles puede participar como propietario en una relación identificadora con otro conjunto de entidades débiles. También es posible tener un conjunto de entidades débiles con más de un conjunto de entidades identificadoras. Una entidad débil en concreto podría ser identificada por una combinación de entidades, una de cada conjunto de entidades identificadoras. La clave primaria de la entidad débil consistiría de la unión de las claves primarias de los conjuntos de entidades identificadoras más el discriminante del conjunto de entidades débiles.

Un conjunto de entidades débiles se indica en los diagramas E-R mediante un rectángulo dibujado con una línea doble y la correspondiente relación de identificación mediante un rombo dibujado con línea doble. En la Figura 2.16, el conjunto de entidades débiles *pago* es dependiente del conjunto de entidades fuertes *préstamo* a través del conjunto de relaciones *pago-préstamo*.

La figura ilustra también el uso de líneas dobles para indicar *participación total*; la participación del conjunto de entidades (débiles) *pago* en la relación *pago-préstamo* es total, significando que cada pago debe estar relacionando a través de *pago-préstamo* con alguna cuenta. Finalmente, la flecha desde *pago-préstamo* a *préstamo* indica que cada pago es para un único préstamo. El discriminante del conjunto de entidades débiles también está subrayado, pero con un línea discontinua, en lugar de una continua.

En algunos casos, el diseñador de la base de datos puede elegir expresar un conjunto de entidades débiles como un atributo compuesto multivalorado del conjunto de entidades propietarias. En el ejemplo, esta alternativa requeriría que el conjunto de entidades *préstamo* tuviera un atributo compuesto y multivalorado *pago*, que constara de *número-pago*, *fecha-pago* e *importe-pago*. Un conjunto de entidades débiles se puede modelar más adecuadamente como un atributo si sólo participa en la relación identificadora y si tiene pocos atributos. Alternativamente, una representación de conjunto de entidades débiles será más adecuada para modelar una situación en la que el conjunto participe en otras relaciones además de la relación identificadora y donde el conjunto de entidades débiles tenga muchos atributos.

Como otro de un conjunto de entidades que se puede modelar como un conjunto de entidades débiles considérense las ofertas de asignaturas en una universidad. La misma asignatura se puede ofrecer en diferentes cursos y dentro de un curso puede haber varios grupos para la misma asignatura. Así, se crea un conjunto de entidades débiles *oferta-asignatura*, que depende existencialmente de *asignatura*; las diferentes ofertas de la misma asignatura se identifican por un *curso* y un *número-grupo*, que forma un discriminante pero no una clave primaria.

2.7. CARACTERÍSITCAS DEL MODELO E-R EXTENDIDO

Aunque los conceptos básicos de E-R pueden modelar la mayoría de las características de las bases de datos, algunos aspectos de una base de datos pueden ser más adecuadamente expresados mediante ciertas extensiones del modelo E-R básico. En este apartado se discuten las características E-R extendidas de especialización, generalización, conjuntos de entidades de nivel más alto y más bajo, herencia de atributos y agregación.

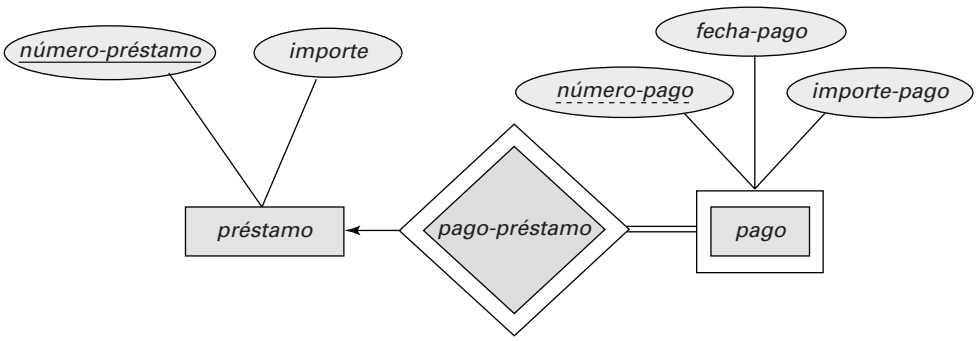

FIGURA 2.16. Diagrama E-R con un conjunto de entidades débiles.

2.7.1. Especialización

Un conjunto de entidades puede incluir subgrupos de entidades que se diferencian de alguna forma de las otras entidades del conjunto. Por ejemplo, un subconjunto de entidades en un conjunto de entidades puede tener atributos que no son compartidos por todas las entidades del conjunto de entidades. El modelo E-R proporciona una forma de representación de estos grupos de entidades distintos.

Considérese el conjunto de entidades *persona* con atributos *nombre*, *calle* y *ciudad*. Una persona puede clasificarse además como:

- *cliente*
- *empleado*

Cada uno de estos tipos de persona se describen mediante un conjunto de atributos que incluyen los atributos del conjunto de entidades *persona* más otros posibles atributos adicionales. Por ejemplo, las entidades *cliente* se pueden describir además mediante el atributo *id-cliente*, mientras que las entidades *empleado* se pueden describir además mediante los atributos *id-empleado* y *sueldo*. El proceso de designación de subgrupos dentro de un conjunto de entidades se denomina **especialización**. La especialización de *persona* permite distinguir entre las personas basándose en si son empleados o clientes.

Como otro ejemplo supóngase que el banco desea dividir las cuentas en dos categorías: cuentas corrientes y cuentas de ahorro. Las cuentas de ahorro necesitan un saldo mínimo, pero el banco establece diferentes tasas de interés a cada cliente, ofreciendo mejores tasas a los clientes favorecidos. Las cuentas corrientes tienen una tasa fija de interés, pero permiten los descubiertos; el importe de descubierto de una cuenta corriente se debe registrar.

El banco podría crear dos especializaciones de *cuenta*, denominadas *cuenta-ahorro* y *cuenta-corriente*. Como se vio anteriormente, las entidades *cuenta* se describen por los atributos *número-cuenta* y *saldo*. El conjunto de entidades *cuenta-ahorro* tendría todos los atributos de *cuenta* y un atributo adicional denominado *tasa-interés*. El conjunto de entidades *cuenta-corriente* tendría todos los atributos de *cuenta* y un atributo adicional *importe-descubierto*.

Se puede aplicar repetidamente la especialización para refinar el esquema de diseño. Por ejemplo, los empleados del banco se pueden clasificar en uno de los siguientes:

- *oficial*
- *cajero*
- *secretaria*

Cada uno de estos tipos de empleado se describe por un conjunto de atributos que incluye todos los atributos del conjunto de entidades *empleado* más otros adicionales. Por ejemplo, las entidades *oficial* se puede describir por el atributo *número-despacho*, las entidades *cajero* por los atributos *número-sección* y *horas-semana*, y las entidades *secretaria* por el atributo *horas-semana*. Además, las entidades *secretaria* pueden participar en una relación *secretaria-de*, que identifica al empleado ayudado por una secretaria.

Un conjunto de entidades se puede especializar por más de una característica distintiva. En el ejemplo, la característica distintiva entre entidades *empleado* es el trabajo que realiza el empleado. Otra especialización coexistente podría estar basada en si la persona es un trabajador temporal o fijo, resultado en los conjuntos de entidades *empleado-temporal* y *empleado-fijo*. Cuando se forma más de una especialización de un conjunto de entidades, una entidad en particular puede pertenecer a varias especializaciones. Por ejemplo, una empleada dada puede ser una empleada temporal y secretaria.

En términos de un diagrama E-R, la especialización se representa mediante un componente *triangular* etiquetado ES, como se muestra en la Figura 2.17. La etiqueta ES representa, por ejemplo, que un cliente «es» una persona. La relación ES se puede llamar también relación **superclase-subclase**. Los conjuntos de entidades de nivel más alto y más bajo se representan como conjuntos de entidades regulares, es decir, como rectángulos que contienen el nombre del conjunto de entidades.

2.7.2. Generalización

El refinamiento a partir de un conjunto de entidades inicial en sucesivos niveles de subgrupos de entidades representa un proceso de diseño **descendente** en el que las distinciones se hacen explícitas. El proceso de diseño puede ser también de una forma **ascendente**, en el que varios conjuntos de entidades se sintetizan en un conjunto de entidades de nivel más alto basado en características comunes. El diseñador de la base de datos puede haber identificado primero el conjunto de entidades *cliente* con los atributos *nombre*, *calle*, *ciudad* e *id-cliente*, y el conjunto de entidades *empleado* con los atributos *nombre*, *calle*, *ciudad*, *id-empleado* y *sueldo*.

Hay similitudes entre el conjunto de entidades *cliente* y el conjunto de entidades *empleado* en el sentido de que tienen varios atributos en común. Esta similitud se puede expresar mediante la **generalización**, que es una relación contenedora que existe entre el conjunto de entidades de *nivel más alto* y uno o más conjuntos de entidades de *nivel más bajo*. En el ejemplo, *persona* es el conjunto de entidades de nivel más alto y los conjuntos de entidades *cliente* y *empleado* son de nivel más bajo. Los conjuntos de entidades de nivel más alto y nivel más bajo también se pueden llamar **superclase** y **subclase**, respectivamente. El conjunto de entidades *persona* es la superclase de las subclases *cliente* y *empleado*.

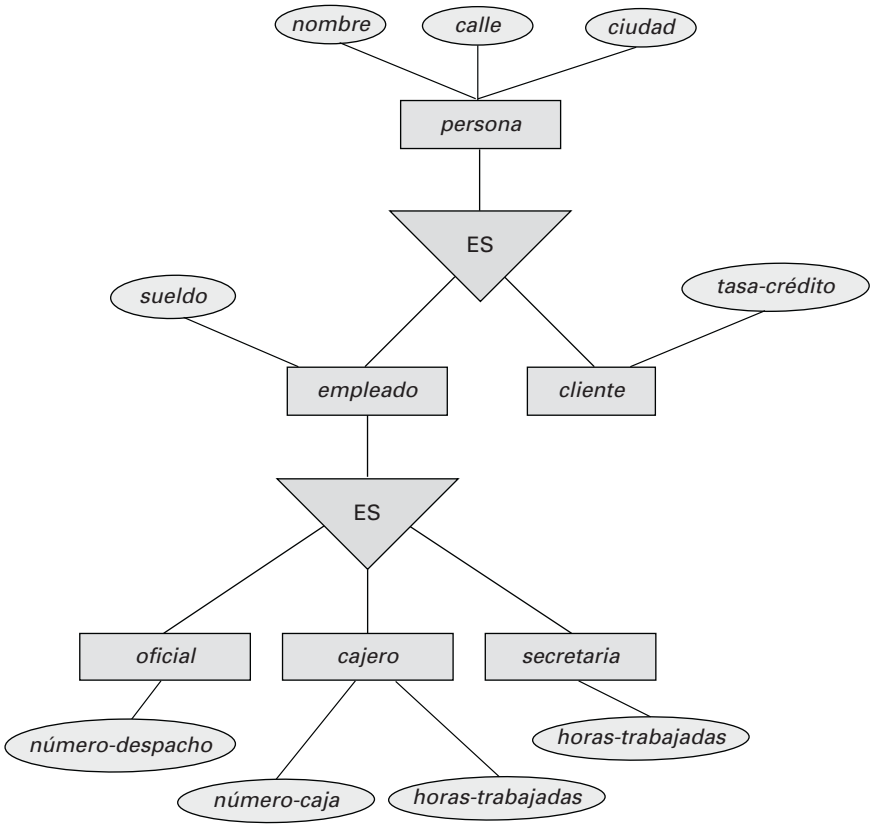

FIGURA 2.17. Especialización y generalización.

Para todos los propósitos prácticos, la generalización es una inversión simple de la especialización. Se aplicarán ambos procesos en combinación en el curso del diseño del esquema E-R para una empresa. En términos del propio diagrama E-R no se distingue entre especialización y generalización. Los niveles nuevos de representación de entidades serán distinguidos (especialización) o sintetizados (generalización) cuando el esquema de diseño llegue a expresar completamente la aplicación de base de datos y los requisitos de uso de la base de datos. Las diferencias entre los dos enfoques se pueden caracterizar mediante su punto de partida y el objetivo global.

La especialización parte de un conjunto de entidades simple; enfatiza las diferencias entre las entidades dentro del conjunto mediante la creación de distintos conjuntos de entidades de nivel más bajo. Estos conjuntos de entidades de nivel más bajo pueden tener atributos, o pueden participar en relaciones que no se aplican a todas las entidades del conjunto de entidades de nivel más alto. Realmente, la razón de que el diseñador aplique la especialización es representar tales características diferentes. Si *cliente* y *empleado* no tuvieran cada una atributos únicos que no tuvieran las entidades *persona* en la que participan, no habría necesidad de especializar el conjunto de entidades *persona*.

La generalización procede de observar que varios conjuntos de entidades que comparten algunas características comunes (se describen mediante los mismos atributos y participan en los mismos conjuntos de relaciones). Basada en sus similitudes, la generalización sintetiza estos conjuntos de entidades en uno solo, el conjunto de entidades de nivel más alto. La generalización se usa para resaltar las similitudes entre los conjuntos de entidades de nivel más bajo y para ocultar las diferencias; también permite economizar la representación para que los atributos compartidos no estén repetidos.

2.7.3. Herencia de atributos

Una propiedad crucial de las entidades de nivel más alto y más bajo creadas mediante especialización y generalización es la **herencia de atributos**. Los atributos de los conjuntos de entidades de nivel más alto se dice que son **heredados** por los conjuntos de entidades de nivel más bajo. Por ejemplo, *cliente* y *empleado* heredan los atributos de *persona*. Así, *cliente* se describe mediante sus atributos *nombre, calle y ciudad* y adicionalmente por el atributo *id-cliente*; *empleado* se describe mediante sus atributos *nombre, calle y ciudad* y adicionalmente por los atributos *id-empleado* y *sueldo*.

Un conjunto de entidades de nivel más bajo (o subclase) también hereda la participación en los conjuntos

de relaciones en los que su entidad de nivel más alto (o superclase) participa. Ambos conjuntos de entidades *oficial*, *cajero* y *secretaria* participan en el conjunto de relaciones *trabaja-para*. La herencia de atributos se aplica en todas las capas de los conjuntos de entidades de nivel más bajo. Los conjuntos de entidades anteriores pueden participar cualquier relación en que participe el conjunto de entidades *persona*.

Si se llega a una porción dada de un modelo E-R mediante especialización o generalización, el resultado es básicamente el mismo:

- Un conjunto de entidades de nivel más alto con atributos y relaciones que se aplican a todos los conjuntos de entidades de nivel más bajo.
- Conjuntos de entidades de nivel más bajo con características distintivas que se aplican sólo en un conjunto de entidades particular.

En lo que sigue, aunque a menudo se hará referencia sólo a la generalización, las propiedades que se discuten pertenecen a ambos procesos.

En la Figura 2.17 se describe una **jerarquía** de conjuntos de entidades. En la figura, *empleado* es un conjunto de entidades de nivel más bajo de *persona* y un conjunto de entidades de nivel más alto de los conjuntos de entidades *oficial*, *cajero* y *secretaria*. En una jerarquía, un conjunto de entidades dado puede estar implicado como un conjunto de entidades de nivel más bajo sólo en una única relación ES. Si un conjunto de entidades es un conjunto de entidades de nivel más bajo en más de una relación ES, entonces el conjunto de entidades tiene **herencia múltiple**, y la estructura resultante se denomina *retículo*.

2.7.4. Restricciones sobre las generalizaciones

Para modelar una empresa más exactamente, el diseñador de la base de datos puede elegir colocar ciertas restricciones en una generalización particular. Un tipo de restricción implica determinar qué entidades pueden ser miembros de un conjunto de entidades de nivel más bajo dado. Tales relaciones de miembros pueden ser algunas de los siguientes:

- **Definido por condición.** En los conjuntos de entidades de nivel más bajo, la relación miembro se evalúa en función de si una entidad satisface o no una condición explícita o predicado. Por ejemplo, asúmase que el conjunto de entidades de nivel más alto *cuenta* tiene el atributo *tipo-cuenta*. Todas las entidades *cuenta* se evalúan según la definición del atributo *tipo-cuenta*. Sólo aquellas entidades que satisfagan la condición *tipo-cuenta* = «cuenta de ahorro» podrán pertenecer al conjunto de entidades de nivel más bajo *cuenta-ahorro*. Todas las entidades que satisfagan la condición *tipo-cuenta* = «cuenta corriente» estarán incluidas en *cuenta-corriente*. Como todas las entidades de nivel más bajo se evalúan en función del mismo atributo (en este caso, *tipo-cuenta*), este tipo de generalización se denomina **definido por atributo**.
- **Definido por el usuario.** Los conjuntos de entidades de nivel más bajo definidos por el usuario no están restringidos mediante una condición de miembro; en cambio, las entidades se asignan a un conjunto de entidades dado por el usuario de la base de datos. Por ejemplo, asúmase que, después de tres meses de empleo, se asignan los empleados del banco a uno de los cuatro grupos de trabajo. Los grupos se representan, por tanto, como cuatro conjuntos de entidades de nivel más bajo del conjunto de entidades de nivel más alto *empleado*. Un empleado dado no se asigna a una entidad grupo automáticamente en términos de una condición que lo defina explícitamente. En su lugar, la asignación al grupo se hace de forma individual por el usuario a cargo de la decisión. Las asignación se implementa mediante una operación que añade una entidad a un conjunto de entidades.

Un segundo tipo de restricciones se define según si las entidades pueden pertenecer a más de un conjunto de entidades de nivel más bajo en una generalización simple. Los conjuntos de entidades de nivel más bajo pueden ser uno de los siguientes:

- **Disjunto.** Una *restricción sobre el carácter disjunto* requiere que una entidad no pertenezca a más de un conjunto de entidades de nivel más bajo. En el ejemplo, una entidad *cuenta* puede satisfacer sólo una condición para el atributo *tipo-cuenta*; una entidad puede ser bien una cuenta de ahorro o bien una cuenta corriente, pero no ambas cosas a la vez.
- **Solapado.** En las *generalizaciones solapadas*, la misma entidad puede pertenecer a más de un conjunto de entidades de nivel más bajo en una generalización simple. Como ilustración, tomando el ejemplo del grupo de trabajo del empleado, asúmase que ciertos directores participen en más de un grupo de trabajo. Un empleado dado puede, por lo tanto, aparecer en más de uno de los conjuntos de entidades grupo que son conjuntos de entidades de nivel más bajo de *empleado*. Así, la generalización es solapada.

  Como otro ejemplo, supóngase la generalización aplicada a los conjuntos de entidades *cliente* y *empleado* conduce a un conjunto de entidades de nivel más alto *persona*. La generalización está solapada si un empleado también puede ser un cliente.

La entidad de nivel más bajo solapada es el caso predeterminado; la restricción sobre el carácter disjunto se debe colocar explícitamente en una generali-

zación (o especialización). Se puede identificar una restricción sobre el carácter disjunto en un diagrama E-R añadiendo la palabra *disjunto* en el símbolo del triángulo.

Una restricción final, la **restricción de completitud** en una generalización o especialización, especifica si un conjunto de entidades de nivel más alto debe pertenecer o no a al menos a uno de los conjuntos de entidades de nivel más bajo en una generalización/especialización. Esta restricción puede ser una de las siguientes:

- **Generalización** o **especialización total.** Cada entidad de nivel más alto debe pertenecer a un conjunto de entidades de nivel más bajo.
- **Generalización** o **especialización parcial.** Algunas entidades de nivel más alto pueden no pertenecer a algún conjunto de entidades de nivel más bajo.

La generalización parcial es la predeterminada. Se puede especificar una generalización total en un diagrama E-R usando una línea doble para conectar el rectángulo que representa el conjunto de entidades de nivel más alto con el símbolo del triángulo (esta notación es similar a la notación de participación total en una relación).

La generalización de *cuenta* es total: todas las entidades *cuenta* deben ser o bien cuentas de ahorro o bien cuentas corrientes. Debido a que el conjunto de entidades de nivel más alto alcanzado a través de la generalización está generalmente compuesta únicamente por aquellas entidades del conjunto de entidades de nivel más bajo, la restricción de completitud para un conjunto de entidades de nivel más alto generalizado es habitualmente total. Cuando la restricción es parcial, la entidad de nivel más alto no aparece necesariamente en el conjunto de entidades de nivel más bajo. Los conjuntos de entidades grupo de trabajo ilustran una especialización parcial. Como los empleados se asignan a grupos sólo después de llevar tres meses en el trabajo, algunas entidades *empleado* pueden no ser miembros de ningún conjunto de entidades grupo de nivel más bajo.

Los conjuntos de entidades equipo se pueden caracterizar más completamente como una especialización de *empleado* parcial y solapada. La generalización de *cuenta-corriente* y *cuenta-ahorro* en *cuenta* es una generalización total y disjunta. Las restricciones de completitud y sobre el carácter disjunto, sin embargo, no dependen una de la otra. Los patrones de restricciones pueden ser también parcial-disjunta y total-solapada.

Se puede ver que ciertos requisitos de inserción y borrado son consecuencia de las restricciones que se aplican a una generalización o especialización dada. Por ejemplo, cuando se coloca una restricción de completitud total, una entidad insertada en un conjunto de entidades de nivel más alto se debe insertar en al menos uno de los conjuntos de entidades de nivel más bajo. Con una restricción de definición por condición, todas las entidades de nivel más alto que satisfacen la condición se deben insertar en el conjunto de entidades de nivel más bajo. Finalmente, una entidad que se borra de un conjunto de entidades de nivel más alto, también se debe borrar de todos los conjuntos de entidades de nivel más bajo asociados a los que pertenezca.

2.7.5. Agregación

Una limitación del modelo E-R es que no resulta posible expresar relaciones entre relaciones. Para ilustrar la necesidad de tales construcciones considérese la relación ternaria *trabaja-en*, que se vio anteriormente, entre *empleado*, *sucursal* y *trabajo* (véase la Figura 2.13). Supóngase ahora que se desean registrar los directores para las tareas realizadas por un empleado en una sucursal; es decir, se desean registrar directores por combinaciones (*empleado*, *sucursal*, *trabajo*). Asúmase que existe una entidad *director*.

Una alternativa para representar esta relación es crear una relación cuaternaria *dirige* entre *empleado*, *sucursal*, *trabajo* y *director* (se necesita una relación cuaternaria; una relación binaria entre *director* y *empleado* no permitiría representar las combinaciones [*sucursal*, *trabajo*] de un empleado que están dirigidas por un director). Al usar los constructores básicos del modelado E-R se obtiene el diagrama E-R de la Figura 2.18 (por simplicidad se han omitido los atributos).

Parece que los conjuntos de relaciones *trabaja-en* y *dirige* se pueden combinar en un único conjunto de relaciones. No obstante, no se deberían combinar, dado que algunas combinaciones *empleado*, *sucursal*, *trabajo* puede que no tengan director.

Hay información redundante en la figura resultante, ya que cada combinación *empleado*, *sucursal*, *trabajo* en *dirige* también lo está en *trabaja-en*. Si el director fuese un valor en lugar de una entidad *director*, se podría hacer que *director* fuese un atributo multivalorado de la relación *trabaja-en*. Pero esto implica que es más difícil (tanto lógicamente como en coste de ejecución) encontrar, por ejemplo, los triples empleado-sucursal-trabajo de los que un director es responsable. Como el director es una entidad *director*, se descarta esta alternativa en cualquier caso.

La mejor forma de modelar una situación como ésta es usar la agregación. La **agregación** es una abstracción a través de la cual las relaciones se tratan como entidades de nivel más alto. Así, para este ejemplo, se considera el conjunto de relaciones *trabaja-en* (que relaciona los conjuntos de entidades *empleado*, *sucursal* y *trabajo*) como un conjunto de entidades de nivel más alto denominado *trabaja-en*. Tal conjunto de entidades se trata de la misma forma que cualquier otro conjunto de entidades. Se puede crear entonces una relación binaria *dirige* entre *trabaja-en* y *director* para representar

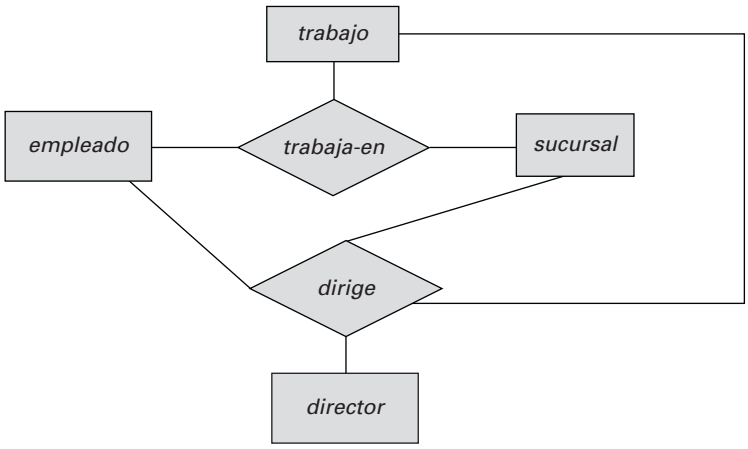

FIGURA 2.18. Diagrama E-R con relaciones redundantes.

quién dirige las tareas. En la Figura 2.19 se muestra una notación para la agregación que se usa habitualmente para esta situación.

2.7.6. Notaciones E-R alternativas

La Figura 2.20 resume el conjunto de símbolos que hemos usado en los diagramas E-R. No hay ningún estándar universal para la notación de los diagramas E-R y diferentes libros y diferente software de diagramas E-R usan notaciones diferentes; la Figura 2.21 indica alguna de las notaciones alternativas que se usan ampliamente. Un conjunto de entidades se puede representar como un cuadro con el nombre fuera, y los atributos listados unos debajo de otros dentro del cuadro. Los atributos clave primaria se indican listándolos en la parte superior, con una línea separándolos de los otros atributos.

Las restricciones de cardinalidad se pueden indicar de varias formas como se muestra en la Figura 2.21. Las etiquetas * y 1 en los arcos que salen de las relaciones se usan a menudo para denotar relaciones varios a varios, uno a uno y varios a uno como se muestra en la figura. En otra notación alternativa de la figura los conjuntos de relaciones se representan por líneas entre conjuntos de entidades sin rombos; sólo se pueden modelar de esta forma las relaciones binarias. Las restricciones de cardinalidad en esta notación se muestran por la notación «pata de gallo», como en la figura.

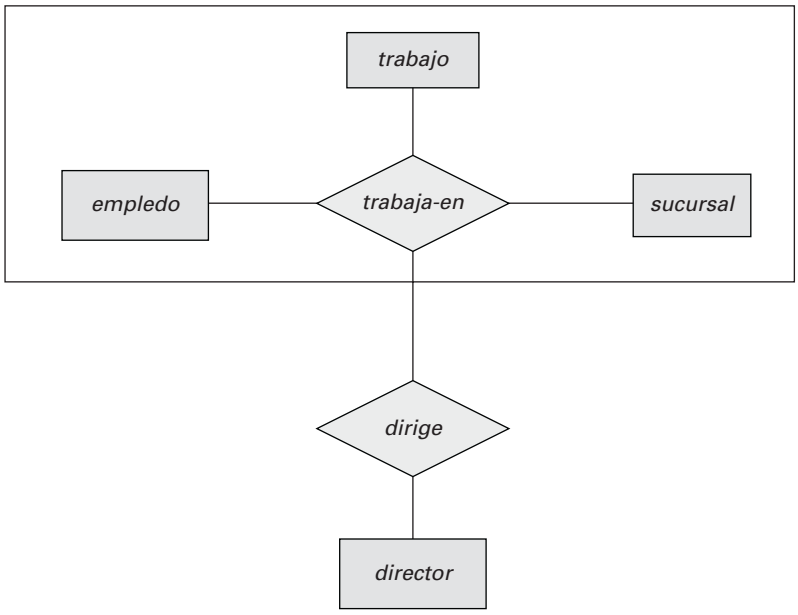

FIGURA 2.19. Diagrama E-R con agregación.

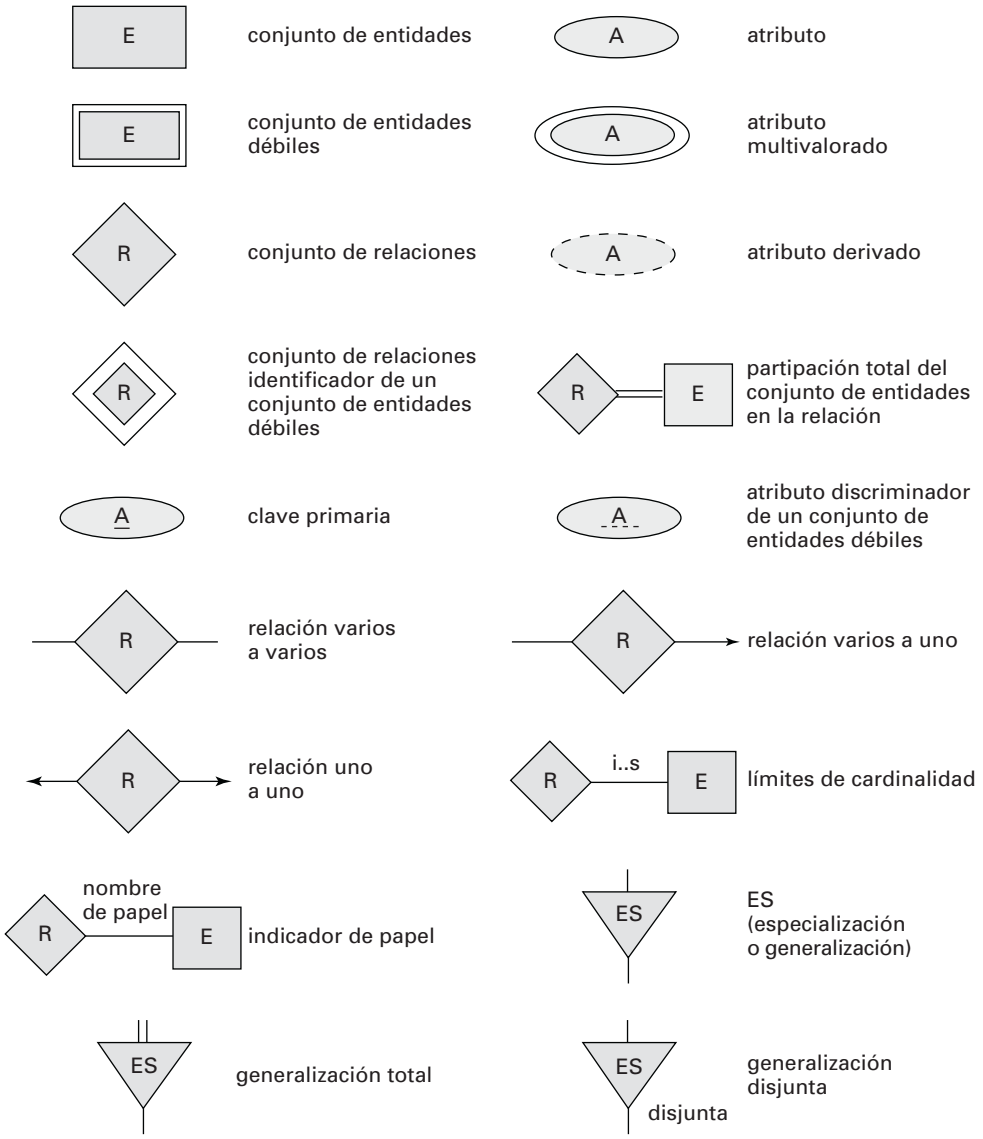

FIGURA 2.20. Símbolos usados en la notación E-R.

2.8. DISEÑO DE UN ESQUEMA DE BASE DE DATOS E-R

El modelo de datos E-R da una flexibilidad sustancial en el diseño de un esquema de bases de datos para modelar una empresa dada. En este apartado se considera cómo un diseñador de bases de datos puede seleccionar entre el amplio rango de alternativas. Entre las decisiones que se toman están las siguientes:

- Si se usa un atributo o un conjunto de entidades para representa un objeto (discutido anteriormente en el Apartado 2.2.1)
- Si un concepto del mundo real se expresa más exactamente mediante un conjunto de entidades o mediante un conjunto de relaciones (Apartado 2.2.2)
- Si se usa una relación ternaria o un par de relaciones binaras (Apartado 2.2.3)
- Si se usa un conjunto de entidades fuertes o débiles (Apartado 2.6); un conjunto de entidades fuertes y sus conjuntos de entidades débiles dependientes se pueden considerar como un «objeto» en la base de datos, debido a que la existencia de las entidades débiles depende de la entidad fuerte
- Si el uso de la generalización (Apartado 2.7.2) es apropiado; la generalización, o una jerarquía de relaciones ES, contribuye a la modularidad por permitir que los atributos comunes de conjuntos de entidades similares se representen en un único lugar en un diagrama E-R
- Si el uso de la agregación (Apartado 2.7.5) es apropiado; la agregación agrupa una parte de un dia-

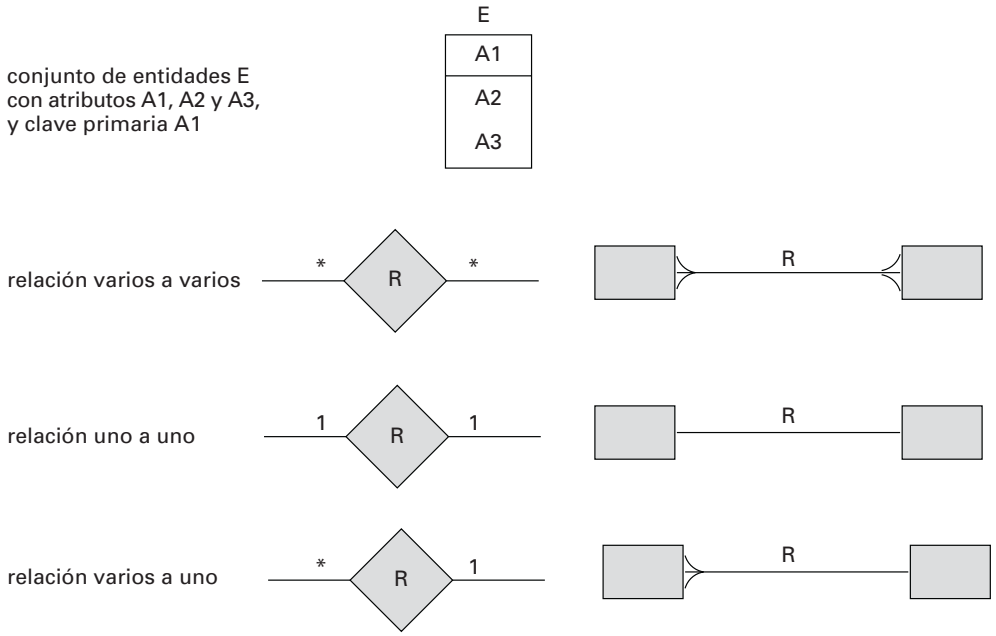

FIGURA 2.21. Notaciones E-R alternativas.

grama E-R en un único conjunto de entidades, permitiendo tratar el conjunto de entidades de la agregación como una unidad única sin importar los detalles de su estructura interna.

Se verá que el diseñador de bases de datos necesita un buen entendimiento de la empresa que se modela para tomar estas decisiones.

2.8.1. Fases de diseño

Un modelo de datos de alto nivel sirve al diseñador de la base de datos para proporcionar un marco conceptual en el que especificar de forma sistemática los requisitos de datos de los usuarios de la base de datos que existen, y cómo se estructurará la base de datos para completar estos requisitos. La fase inicial del diseño de bases de datos, por tanto, es caracterizar completamente las necesidades de datos esperadas por los usuarios de la base de datos. El resultado de esta fase es una *especificación de requisitos del usuario*.

A continuación, el diseñador elige un modelo de datos y, aplicando los conceptos del modelo de datos elegido, traduce estos requisitos a un esquema conceptual de la base de datos. El esquema desarrollado en esta fase de **diseño conceptual** proporciona una visión detallada del desarrollo. Debido a que sólo se ha estudiado el modelo E-R hasta ahora, se usará éste para desarrollar el esquema conceptual. En términos del modelo E-R, el esquema especifica todos los conjuntos de entidades, conjuntos de relaciones, atributos y restricciones de correspondencia. El diseñador revisa el esquema para confirmar que todos los requisitos de datos se satisfacen realmente y no hay conflictos entre sí. También se examina el diseño para eliminar características redundantes. Lo importante en este punto es describir los datos y las relaciones, más que especificar detalles del almacenamiento físico.

Un esquema conceptual completamente desarrollado indicará también los requisitos funcionales de la empresa. En una **especificación de requisitos funcionales** los usuarios describen los tipos de operaciones (o transacciones) que se realizarán sobre los datos. Algunos ejemplos de operaciones son la modificación o actualización de datos, la búsqueda y recuperación de datos específicos y el borrado de datos. En esta fase de diseño conceptual se puede hacer una revisión del esquema para encontrar los requisitos funcionales.

El proceso de trasladar un modelo abstracto de datos a la implementación de la base de datos consta de dos fases de diseño finales. En la **fase de diseño lógico**, el diseñador traduce el esquema conceptual de alto nivel al modelo de datos de la implementación del sistema de base de datos que se usará. El diseñador usa el esquema resultante específico a la base de datos en la siguiente **fase de diseño físico**, en la que se especifican las características físicas de la base de datos. Estas características incluyen la forma de organización de los archivos y las estructuras de almacenamiento interno, que se discutirán en el Capítulo 11.

En este capítulo se tratan sólo los conceptos del modelo E-R usados en la fase de diseño del esquema conceptual. Se ha presentado una breve visión del proceso de diseño de bases de datos para proporcionar un contexto para la discusión del modelo de datos E-R. El diseño de bases de datos recibe un tratamiento completo en el Capítulo 7.

En el Apartado 2.8.2 se aplican las dos fases iniciales de diseño de bases de datos al ejemplo del banco.

Se emplea el modelo de datos E-R para traducir los requisitos de usuario al esquema de diseño conceptual que se describe como un diagrama E-R.

2.8.2. Diseño de base de datos para el banco

Nos centramos ahora en los requisitos de diseño de la base de datos para el banco en más detalle y desarrollamos un diseño más realista, aunque también más complicado, de lo que se ha visto en los ejemplos anteriores. Sin embargo, no se intentará modelar cada aspecto del diseño de la base de datos para un banco; se considerarán sólo unos cuantos aspectos para ilustrar el proceso de diseño de bases de datos.

2.8.2.1. Requisitos de datos

La especificación inicial de los requisitos de usuario se puede basar en entrevistas con los usuarios de la base de datos y en el análisis propio del diseñador del desarrollo. La descripción que surge de esta fase de diseño sirve como base para especificar la estructura conceptual de la base de datos. La siguiente lista describe los principales requisitos del banco:

- El banco está organizado en sucursales. Cada sucursal está ubicada en una ciudad particular y se identifica por un nombre único. El banco supervisa los activos de cada sucursal.
- Los clientes del banco se identifican mediante sus valores de *id-cliente*. El banco almacena cada nombre de cliente, y la calle y ciudad donde viven los clientes. Los clientes pueden tener cuentas y pueden pedir préstamos. Un cliente puede estar asociado con un banquero particular, que puede actuar como responsable de préstamos o banquero personal para un cliente.
- Los empleados del banco se identifican mediante sus valores de *id-empleado*. La administración del banco almacena el nombre y número de teléfono de cada empleado, los nombres de los subordinados del empleado, y el número *id-empleado* del jefe del empleado. El banco también mantiene registro de la fecha de comienzo del contrato del empleado, así como su antigüedad.
- El banco ofrece dos tipos de cuentas: cuentas de ahorro y cuentas corrientes. Las cuentas pueden asociarse a más de un cliente y un cliente puede tener más de una cuenta. Cada cuenta está asignada a un único número de cuenta. El banco mantiene un registro del saldo de cada cuenta y la fecha más reciente en que la cuenta fue accedida por cada cliente que mantiene la cuenta. Además, cada cuenta de ahorro tiene un tipo de interés y para cada cuenta corriente se almacena el descubierto.
- Un préstamo tiene lugar en una sucursal particular y puede estar asociado a uno o más clientes. Un préstamo se identifica mediante un único número de préstamo. Para cada préstamo el banco mantiene registro del importe del préstamo y de los pagos del préstamo. Aunque un número de pago del préstamo no identifica de forma única un pago entre todos los préstamos del banco, un número de pago identifica un pago particular para un préstamo específico. Para cada pago se almacenan la fecha y el importe.

En un desarrollo de un banco real, el banco mantendría información de los abonos y cargos en las cuentas de ahorros y en las cuentas corrientes, igual que se mantiene registro de los pagos para los préstamos. Debido a que los requisitos del modelo para este seguimiento son similares, y para mantener nuestro ejemplo reducido, en este modelo no se mantiene un seguimiento de tales abonos y cargos.

2.8.2.2. Designación de los conjuntos de entidades

La especificación de los requisitos de datos sirve como punto de partida para la construcción de un esquema conceptual para la base de datos. Desde la especificación listada en el Apartado 2.8.2.1 se comienzan a identificar los conjuntos de entidades y sus atributos.

- El conjunto de entidades *sucursal*, con los atributos *nombre-sucursal*, *ciudad-sucursal* y *activo*.
- El conjunto de entidades *cliente*, con los atributos *id-cliente*, *nombre-cliente*, *calle-cliente* y *ciudad-cliente*. Un posible atributo adicional es *nombre-banquero*.
- El conjunto de entidades *empleado*, con los atributos *id-empleado, nombre-empleado, número-teléfono, sueldo y jefe*. Algunas características descriptivas adicionales son el atributo multivalorado *nombre-subordinado*, el atributo base *fecha-comienzo* y el atributo derivado *antigüedad*.
- Dos conjuntos de entidades cuenta —*cuenta-ahorro* y *cuenta-corriente*— con los atributos comunes *número-cuenta* y *saldo*; además, *cuenta-ahorro* tiene el atributo *tipo-interés* y *cuenta-corriente* tiene el atributo *descubierto*.
- El conjunto de entidades *préstamo*, con los atributos *número-préstamo*, *importe* y *sucursal-origen*.
- El conjunto de entidades débiles *pago-préstamo*, con los atributos *número-pago*, *fecha-pago* e *importe-pago*.

2.8.2.3. Designación de los conjuntos de relaciones

Volviendo ahora al esquema de diseño rudimentario del Apartado 2.8.2.2 se especifican los siguientes conjuntos de relaciones y correspondencia de cardinalidades:

- *prestatario*, un conjunto de relaciones varios a varios entre *cliente* y *préstamo*.

- *préstamo-sucursal*, un conjunto de relaciones varios a uno que indica la sucursal en que se ha originado un préstamo. Nótese que este conjunto de relaciones reemplaza al atributo *sucursal-origen* del conjunto de entidades *préstamo*.
- *pago-préstamo*, un conjunto de relaciones uno a varios de *préstamo* a *pago*, que documenta que se ha realizado un pago de un préstamo.
- *impositor*, con el atributo de relación *fecha-acceso*, un conjunto de relaciones varios a varios entre *cliente* y *cuenta*, indicando que un cliente posee una cuenta.
- *banquero-consejero*, con el atributo de relación *tipo*, un conjunto de relaciones varios a uno que expresa que un cliente puede ser aconsejado por un empleado del banco, y que un empleado del banco puede aconsejar a uno o más clientes. Nótese que este conjunto de relaciones ha reemplazado al atributo *nombre-banquero* del conjunto de entidades *cliente*.
- *trabaja-para*, un conjunto de relaciones entre entidades *empleado* con papeles que indican *jefe* y *trabajador*; la correspondencia de cardinalidades expresa que un empleado trabaja para un único jefe, y que un jefe supervisa uno o más empleados. Nótese que este conjunto de relaciones reemplaza el atributo *jefe* de *empleado*.

2.8.2.4. *Diagrama E-R*

Conforme a lo discutido en el Apartado 2.8.2.3 se presenta ahora el diagrama E-R completo para el ejemplo del banco. En la Figura 2.22 se muestra la representación

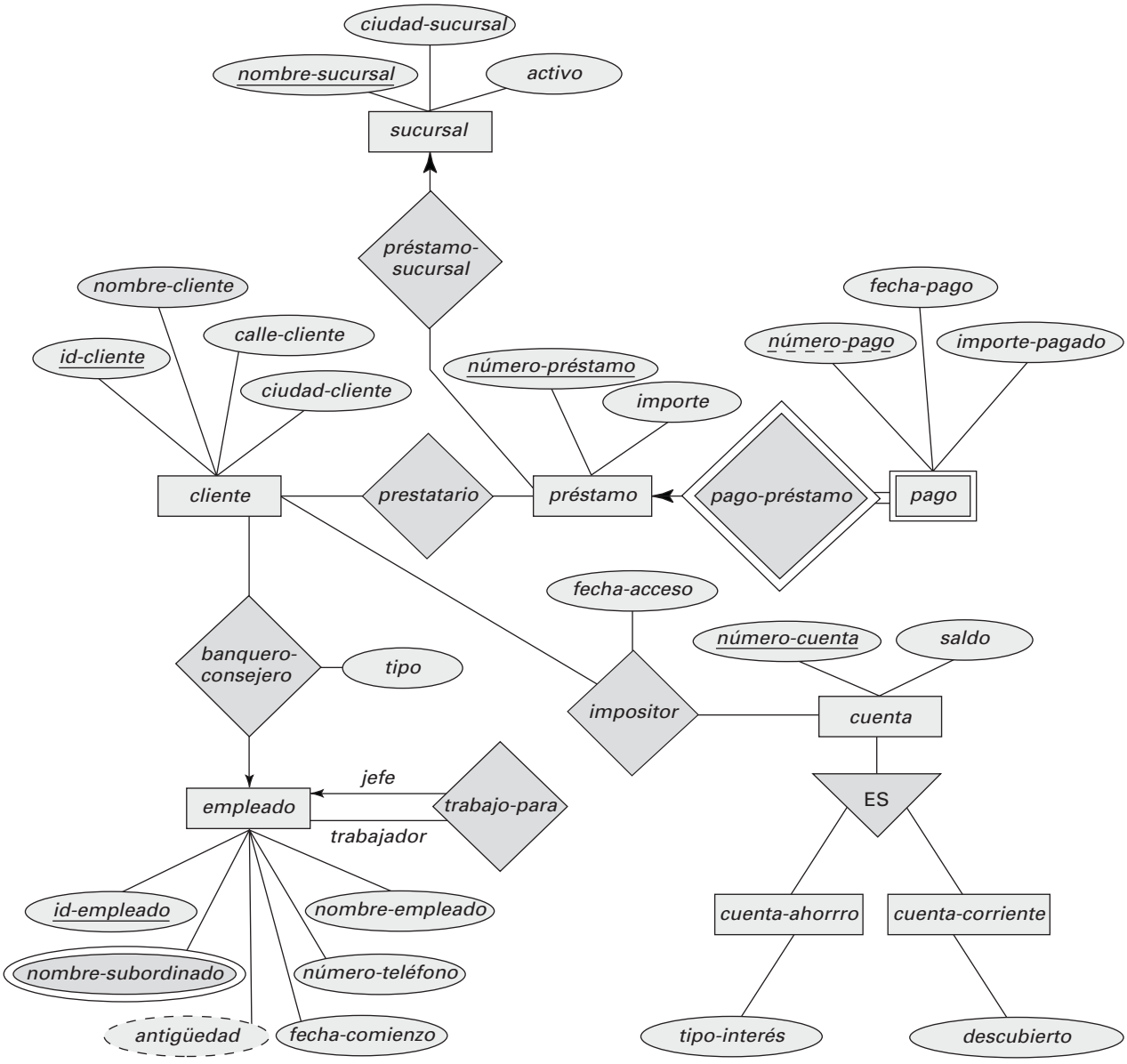

FIGURA 2.22. Diagrama E-R para un banco.

completa de un modelo conceptual de un banco, expresada en términos de los conceptos E-R. El diagrama incluye los conjuntos de entidades, atributos, conjuntos de relaciones, y correspondencia de cardinalidades alcanzados a través del proceso de diseño de los Apartados 2.8.2.1 y 2.8.2.2, y refinados en el Apartado 2.8.2.3.

2.9. REDUCCIÓN DE UN ESQUEMA E-R A TABLAS

Una base de datos que se ajusta a un esquema de bases de datos E-R se puede representar por una colección de tablas. Para cada conjunto de entidades de la base de datos y para cada conjunto de relaciones de la base de datos hay una única tabla a la que se asigna el nombre del conjunto de entidades o del conjunto de relaciones correspondiente. Cada tabla tiene varias columnas, cada una de las cuales tiene un nombre único.

Los modelos E-R y el de bases de datos relacionales son representaciones abstractas y lógicas de empresas del mundo real. Debido a que los dos modelos emplean principios de diseño similares, se puede convertir un diseño E-R en un diseño relacional. Convertir una representación de bases de datos de un diagrama E-R a un formato de tablas es la base para la derivación de un diseño de bases de datos relacional desde un diagrama E-R. Aunque existen diferencias importantes entre una relación y una tabla, una relación se puede considerar informalmente como una tabla de valores.

En este apartado se describe cómo se puede representar un esquema E-R mediante tablas; y en el Capítulo 3 se muestra cómo generar un esquema de bases de datos relacional a partir de un esquema E-R.

Las restricciones especificadas en un diagrama E-R, tales como las claves primarias y las restricciones de cardinalidad, se corresponden con restricciones sobre las tablas generadas a partir del diagrama E-R. Se proporcionan más detalles sobre esta correspondencia en el Capítulo 6 después de describir cómo especificar restricciones sobre tablas.

2.9.1. Representación tabular de los conjuntos de entidades fuertes

Sea E un conjunto de entidades fuertes con los atributos descriptivos $a_1, a_2, \ldots, a_n$. Esta entidad se representa mediante una tabla llamada E con n columnas distintas, cada una de las cuales corresponde a uno de los atributos de E. Cada fila de la tabla corresponde a una entidad del conjunto de entidades E. (En los apartados 2.9.4 y 2.9.5 se describe cómo manjear los atributos compuestos y multivalorados.)

Como ilustración considérese el conjunto de entidades *préstamo* del diagrama E-R mostrado en la Figura 2.8. Este conjunto de entidades tiene dos atributos: *número-préstamo* e *importe*. Se representa este conjunto de entidades mediante una tabla llamada *préstamo*, con dos columnas, como se muestra en la Figura 2.23. La fila

(P-17,1.000)

número-préstamo	*importe*
P-11	900
P-14	1.500
P-15	1.500
P-16	1.300
P-17	1.000
P-23	2.000
P-93	500

FIGURA 2.23. La tabla *préstamo*.

de la tabla *préstamo* significa que el número de préstamo P-17 tiene un importe de préstamo de 1.000 €. Se puede añadir una nueva entidad a la base de datos insertando una fila en una tabla. También se pueden borrar o modificar las filas.

D_1 denota el conjunto de todos los números de préstamo y D_2 denota el conjunto de todos los saldos. Cualquier fila de la tabla *préstamo* debe consistir en una tupla (v_1,v_2), donde v_1 es un número de préstamo (es decir, v_1 está en el conjunto D_1) y v_2 es un importe (es decir, v_2 está en el conjunto D_2). En general, la tabla *préstamo* contendrá sólo un subconjunto del conjunto de todas las filas posibles. El conjunto de todas las filas posibles de *préstamo* es el *producto cartesiano* de D_1 y D_2, denotado por

$$D_1 \times D_2$$

En general, si se tiene una tabla de n columnas, se denota el producto cartesiano de $D_1, D_2, \ldots, D_n$ por

$$D_1 \times D_2 \times \ldots \times D_{n-1} \times D_n$$

Como otro ejemplo considérese el conjunto de entidades *cliente* del diagrama E-R mostrado en la Figura 2.8. Este conjunto de entidades tiene los atributos *id-cliente*, *nombre-cliente*, *calle-cliente* y *ciudad-cliente*. La tabla correspondiente a *cliente* tiene cuatro columnas, como se muestra en la Figura 2.24.

2.9.2. Representación tabular de los conjuntos de entidades débiles

Sea A un conjunto de entidades débiles con los atributos $a_1, a_2, \ldots, a_m$. Sea B el conjunto de entidades fuertes del que A depende. Sea la clave primaria de B el conjunto de atributos $b_1, b_2, \ldots, b_n$. Se representa el conjunto de entidades A mediante una tabla llamada A con una columna por cada uno de los atributos del conjunto:

id-cliente	*nombre-cliente*	*calle-cliente*	*ciudad-cliente*
01.928.374	Gómez	Carretas	Cerceda
18.273.609	Abril	Preciados	Valsaín
19.283.746	González	Arenal	La Granja
24.466.880	Pérez	Carretas	Cerceda
32.112.312	Santos	Mayor	Peguerinos
33.557.799	Fernández	Jazmín	León
33.666.999	Rupérez	Ramblas	León
67.789.901	López	Mayor	Peguerinos
96.396.396	Valdivieso	Goya	Vigo

FIGURA 2.24. La tabla *cliente*.

$$\{a_1, a_2,\ldots,a_m\} \cup \{b_1, b_2,\ldots,b_n\}$$

Como ilustración considérese el conjunto de entidades *pago* mostrado en el diagrama E-R de la Figura 2.16. Este conjunto de entidades tiene tres atributos: *número-pago*, *fecha-pago* e *importe-pago*. La clave primaria del conjunto de entidades *préstamo*, de la que *pago* depende, es *número-préstamo*. Así, *pago* se representa mediante una tabla con cuatro columnas etiquetadas con *número-préstamo*, *número-pago*, *fecha-pago* e *importe-pago*, como se describe en la Figura 2.25.

2.9.3. Representación tabular de los conjuntos de relaciones

Sea R un conjunto de relaciones, sean $a_1, a_2,\ldots,a_m$ el conjunto de atributos formados por la unión de las claves primarias de cada uno de los conjuntos de entidades que participan en R, y sean $b_1, b_2,\ldots,b_n$ los atributos descriptivos de R (si los hay). El conjunto de relaciones se representa mediante una tabla llamada R con una columna por cada uno de los atributos del conjunto:

$$\{a_1, a_2,\ldots,a_m\} \cup \{b_1, b_2,\ldots,b_n\}$$

Como ilustración considérese el conjunto de relaciones *prestatario* del diagrama E-R de la Figura 2.8. Este conjunto de relaciones involucra los dos siguientes conjuntos de entidades:

- *cliente*, con la clave primaria *id-cliente*.
- *préstamo*, con la clave primaria *número-préstamo*.

Debido a que el conjunto de relaciones no tiene atributos, la tabla *prestatario* tiene dos columnas etiquetadas *id-cliente* y *número-préstamo*, como se muestra en la Figura 2.22.

2.9.3.1. Redundancia de tablas

Un conjunto de relaciones uniendo un conjunto de entidades débiles con el correspondiente conjunto de entidades fuertes es un caso especial. Como se hizo notar en el Apartado 2.6, estas relaciones son varios a uno y no tienen atributos descriptivos. Además, la clave primaria de un conjunto de entidades débiles incluye la clave primaria del conjunto de entidades fuertes. En el diagrama E-R de la Figura 2.16, el conjunto de entidades débiles *pago* depende del conjunto de entidades fuertes *préstamo* a través del conjunto de relaciones *pago-préstamo*. La clave primaria de *pago* es {*número-préstamo*, *número-pago*} y la clave primaria de préstamo es {*número-préstamo*}. Como *pago-préstamo* no tiene atributos descriptivos, la tabla para *pago-préstamo* tendría dos columnas, *número-préstamo* y *número-pago*. La tabla para el conjunto de entidades *pago* tiene cuatro columnas, *número-préstamo*, *número-pago*, *fecha-pago* e *importe-pago*. Cada combinación (*número-préstamo*, *número-pago*) en *pago-préstamo* también se encontraría en la tabla *pago*, y viceversa. Por tanto, la tabla *pago-préstamo* es redundante. En general, la tabla para el conjunto de relaciones que une un conjunto de entidades débiles con su correspondiente conjunto de entidades fuertes es redundante y no necesita estar presente en una representación tabular de un diagrama E-R.

número-préstamo	*número-pago*	*fecha-pago*	*importe-pago*
P-11	53	7 junio 2001	125
P-14	69	28 mayo 2001	500
P-15	22	23 mayo 2001	300
P-16	58	18 junio 2001	135
P-17	5	10 mayo 2001	50
P-17	6	7 junio 2001	50
P-17	7	17 junio 2001	100
P-23	11	17 mayo 2001	75
P-93	103	3 junio 2001	900
P-93	104	13 junio 2001	200

FIGURA 2.25. La tabla *pago*.

id-cliente	*número-préstamo*
01.928.374	P-11
01.928.374	P-23
24.466.880	P-93
32.112.312	P-17
33.557.799	P-16
55.555.555	P-14
67.789.901	P-15
96.396.396	P-17

FIGURA 2.26. La tabla *prestatario*.

2.9.3.2. Combinación de tablas

Considérese un conjunto *AB* de relaciones varios a uno del conjunto de entidades *A* al conjunto de entidades *B*. Usando el esquema de construcción de tablas descrito previamente se consiguen tres tablas: *A*, *B* y *AB*. Supóngase además que la participación de *A* en la relación es total; es decir, cada entidad *a* en el conjunto de entidades *A* debe participar en la relación *AB*. Entonces se pueden combinar las tablas *A* y *AB* para formar una única tabla consistente en la unión de las columnas de ambas tablas.

Como ilustración considérese el diagrama E-R de la Figura 2.27. La doble línea del diagrama E-R indica que la participación de *cuenta* en *cuenta-sucursal* es total. Así, una cuenta no puede existir sin estar asociada con una sucursal particular. Además, el conjunto de relaciones *cuenta-sucursal* es varios a uno desde *cuenta* a *sucursal*. Por lo tanto, se puede combinar la tabla para *cuenta-sucursal* con la tabla para *cuenta* y se necesitan sólo las dos tablas siguientes:

- *cuenta*, con los atributos *número-cuenta*, *saldo* y *nombre-cuenta*
- *sucursal*, con los atributos *nombre-sucursal, ciudad-sucursal* y *activo*

En el caso de relaciones uno a uno, la tabla del conjunto de relaciones se puede combinar con las tablas de cualquiera de los conjuntos de entidades. Las tablas se pueden combinar incluso si la participación es parcial usando valores nulos; en el ejemplo anterior se usarían valores nulos para el atributo *nombre-sucursal* para las cuentas que no tengan una sucursal asociada.

2.9.4. Atributos compuestos

Los atributos compuestos se manejan creando un atributo separado para cada uno de los atributos componentes; no se crea una columna separada para el propio atributo compuesto. Supóngase que *dirección* es un atributo compuesto del conjunto de entidades *cliente* y que los componentes de *dirección* son *ciudad* y *calle*. La tabla generada de cliente contendría las columnas *calle-dirección* y *ciudad-dirección*; no hay una columna separada para *dirección*.

2.9.5. Atributos multivalorados

Se ha visto que los atributos en un diagrama E-R generalmente se asocian directamente en columnas para las tablas apropiadas. Los atributos multivalorados, sin embargo, son una excepción; para estos atributos se crean tablas nuevas.

Para un atributo multivalorado *M* se crea una tabla *T* con una columna *C* que corresponde a la clave primaria del conjunto de entidades o conjunto de relaciones del que *M* es atributo. Como ilustración considérese el diagrama E-R de la Figura 2.22. El diagrama incluye el atributo multivalorado *nombre-subordinado*. Para este atributo multivalorado se crea una tabla *nombre-subordinado* con columnas *nombres*, referenciando al atributo *nombre-subordinado* de *empleado*, e *id-empleado*, representado la clave primaria del conjunto de entidades *empleado*. Cada subordinado de un empleado se representa como una única fila en la tabla.

2.9.6. Representación tabular de la generalización

Hay dos métodos diferentes para transformar a forma tabular un diagrama E-R que incluya generalización. Aunque la generalización a la que se va a hacer referencia es la de la Figura 2.17, para simplificar esta discusión se incluye sólo la primera capa de los conjuntos de entidades de nivel más bajo —es decir, *empleado* y *cliente*. Se asume que *nombre* es la clave primaria de *persona*.

1. Crear una tabla para el conjunto de entidades de nivel más alto. Para cada conjunto de entidades

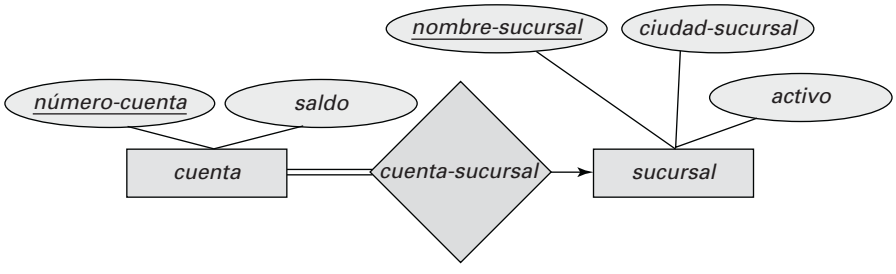

FIGURA 2.27. Diagrama E-R.

de nivel más bajo, crear una tabla que incluya una columna para cada uno de los atributos de ese conjunto de entidades más una columna por cada atributo de la clave primaria del conjunto de entidades de nivel más alto. Así, para el diagrama E-R de la Figura 2.15, se tienen tres tablas:

- *persona*, con atributos *nombre, calle* y *ciudad*
- *empleado*, con atributos *nombre* y *salario*
- *cliente*, con atributos *nombre, límite-crédito*

2. Es posible una representación alternativa si la generalización es disjunta y completa —es decir, si no hay ninguna entidad que sea miembro de dos conjuntos de entidades de menor nivel directamente debajo de un conjunto de entidades de nivel más alto, y si cada entidad del conjunto de entidades de nivel más alto también pertenece a uno de los conjuntos de entidades de nivel más bajo. Aquí no se crea una tabla para el conjunto de entidades de nivel más alto. En su lugar, para cada conjunto de entidades de nivel más bajo se crea una tabla que incluya una columna por cada atributo del conjunto de entidades más una columna por *cada* atributo del conjunto de entidades de nivel más alto. Entonces, para el diagrama E-R de la Figura 2.15 se tienen dos tablas.

   - *empleado*, con atributos *nombre, calle, ciudad* y *sueldo*
   - *cliente*, con atributos *nombre, calle, ciudad* y *límite-crédito*

   Las relaciones *cuenta-ahorro* y *cuenta-corriente* correspondientes a esas tablas tienen *número-cuenta* como clave primaria.

Si se usara el segundo método para una generalización solapada, algunos valores se almacenarían varias veces innecesariamente. Por ejemplo, si una persona es tanto empleado como cliente, los valores de *calle* y *ciudad* se almacenarían dos veces. Si la generalización no fuera completa (es decir, si alguna persona no fuera ni empleado no cliente) entonces se necesitaría una tabla extra *persona* para representarlos.

2.9.7. Representación tabular de la agregación

Transformar a forma tabular un diagrama E-R que incluya agregación es sencillo. Considérese el diagrama de la Figura 2.19. La tabla para el conjunto de relaciones *dirige* entre la agregación de *trabaja-en* y el conjunto de entidades *director* incluye una columna para cada atributo de la clave primaria del conjunto de entidades *director* y del conjunto de relaciones *trabaja-en*. También incluiría una columna para los atributos descriptivos, si los hubiera, del conjunto de relaciones *dirige*. Por tanto, se transforman los conjuntos de relaciones y los conjuntos de entidades dentro de la entidad agregada.

2.10. EL LENGUAJE DE MODELADO UNIFICADO UML (*Unified Modeling Language*)**

Los diagramas entidad-relación ayudan a modelar el componente de representación de datos de un sistema software. La representación de datos, sin embargo, sólo forma parte de un diseño completo de un sistema. Otros componentes son modelos de interacción del usuario con el sistema, especificación de módulos funcionales del sistema y su interacción, etc. El **lenguaje de modelado unificado** (UML, *Unified Modeling Language*) es un estándar propuesto para la creación de especificaciones de varios componentes de un sistema software. Algunas de las partes de UML son:

- **Diagrama de clase.** Un diagrama de clase es similar a un diagrama E-R. Más adelante en este apartado se mostrarán algunas características de los diagramas de clase y cómo se corresponden con los diagramas E-R.
- **Diagrama de caso de uso.** Los diagramas de caso de uso muestran la interacción entre los usuarios y el sistema, en particular los pasos de las tareas que realiza el usuario (tales como prestar dinero o matricularse de una asignatura).
- **Diagrama de actividad.** Los diagramas de actividad describen el flujo de tareas entre varios componentes de un sistema.
- **Diagrama de implementación.** Los diagramas de implementación muestran los componentes del sistema y sus interconexiones tanto en el nivel del componente software como el hardware.

Aquí no se intentará proporcionar un tratamiento detallado de las diferentes partes de UML. Véanse las notas bibliográficas para encontrar referencias de UML. En su lugar se ilustrarán algunas características de UML mediante ejemplos.

La Figura 2.28 muestra varios constructores de diagramas E-R y sus constructores equivalentes de los diagramas de clase UML. Más abajo se describen estos constructores. UML muestra los conjuntos de entidades como cuadros y, a diferencia de E-R, muestra los atributos dentro del cuadro en lugar de como elipses sepa-

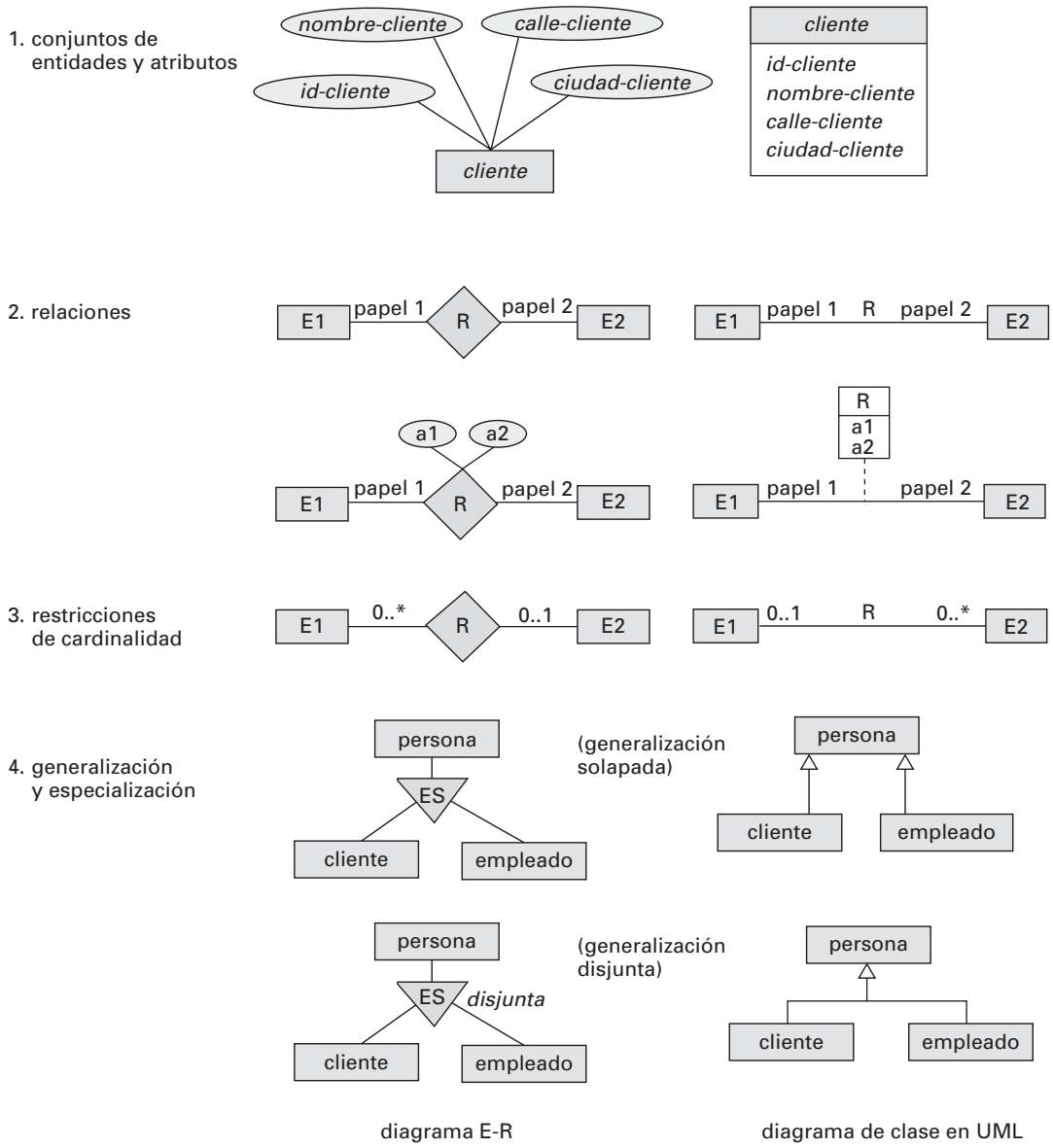

FIGURA 2.28. Símbolos usados en la notación de diagramas de clase UML.

radas. UML modela realmente objetos, mientras que E-R modela entidades. Los objetos son como entidades y tienen atributos, pero además proporcionan un conjunto de funciones (denominadas métodos) que se pueden invocar para calcular valores en términos de los atributos de los objetos, o para modificar el propio objeto. Los diagramas de clase pueden describir métodos además de atributos. Los objetos se tratan en el Capítulo 8.

Los conjuntos de relaciones binarias se representan en UML dibujando simplemente una línea que conecte los conjuntos de entidades. Se escribe el nombre del conjunto de relaciones adyacente a la línea. También se puede especificar el papel que juega un conjunto de entidades en un conjunto de relaciones escribiendo el nombre del papel en un cuadro, junto con los atributos del conjunto de relaciones, y conectar el cuadro con una línea discontinua a la línea que describe el conjunto de relaciones. Este cuadro se puede tratar entonces como un conjunto de entidades, de la misma forma que una agregación en los diagramas E-R puede participar en relaciones con otros conjuntos de entidades.

La relaciones no binarias no se pueden representar directamente en UML —se deben convertir en relaciones binarias por la técnica que se describió en el Apartado 2.4.3.

Las restricciones de cardinalidad se especifican en UML de la misma forma que en los diagramas E-R, de la forma *i..s*, donde *i* denota el mínimo y *h* el máximo número de relaciones en que puede participar una entidad. Sin embargo, se debería ser consciente que la ubicación de las restricciones es exactamente el inverso de la ubicación de las restricciones en los diagramas E-R, como muestra la Figura 2.28. La restricción 0..* en el lado *E*2 y 0..1 en el lado *E*1 significa que cada entidad

*E*2 puede participar a lo sumo en una relación, mientras que cada entidad *E*1 puede participar en varias relaciones; en otras palabras, la relación es varios a uno de *E*2 a *E*1.

Los valores como 1 o * se pueden escribir en los arcos; el valor 1 sobre un arco se trata equivalentemente como 1..1, mientras que * es equivalente a 0..*.

La generalización y especialización se representan en el diagrama E-R conectando conjuntos de entidades por una línea con un triángulo al final correspondiente al conjunto de entidades más general. Por ejemplo, el conjunto de entidades *persona* es una generalización de *cliente* y *empleado*. Los diagramas UML también pueden representar explícitamente las restricciones de generalizaciones disjuntas y solapadas. La Figura 2.28 muestra generalizaciones disjuntas y solapadas de *cliente* y *empleado* a *persona*. Recuérdese que se la generalización de *cliente* / *empleado* a *persona* es disjunta, y significa que ninguna entidad puede ser a la vez un *cliente* y un *empleado*. Una generalización solapada permite que una persona sea tanto *cliente* como *empleado*.

2.11. RESUMEN

- El modelo de datos **entidad-relación (E-R)** se basa en una percepción del mundo real consistente en un conjunto de objetos básicos llamados **entidades** y en **relaciones** entre esos objetos.
- El modelo está pensado principalmente para el proceso de diseño de la base de datos. Fue desarrollado para facilitar el diseño permitiendo la especificación de un **esquema de la empresa**. Tal esquema representa la estructura lógica general de la base de datos. Esta estructura general se puede expresar gráficamente mediante un **diagrama E-R**.
- Una **entidad** es un objeto que existe y es distinguible de otros objetos. Se expresa la distinción asociando con cada entidad un conjunto de atributos que describen el objeto.
- Una **relación** es una asociación entre diferentes entidades. Un **conjunto de relaciones** es una colección de relaciones del mismo tipo y un **conjunto de entidades** es una colección de entidades del mismo tipo.
- La **correspondencia de cardinalidades** expresa el número de entidades a las que otra entidad se puede asociar a través de un conjunto de relaciones.
- Una **superclave** de un conjunto de entidades es un conjunto de uno o más atributos que, tomados colectivamente, permiten identificar unívocamente una entidad en un conjunto de entidades. Se elige una superclave mínima para cada conjunto de entidades de entre sus superclaves; la superclave mínima se denomina la **clave primaria** del conjunto de entidades. Análogamente, un conjunto de relaciones es un conjunto de uno o más atributos que, tomados colectivamente, permiten identificar unívocamente una relación en un conjunto de relaciones. De igual forma se elige una superclave mínima para cada conjunto de relaciones de entre todas sus superclaves; ésta es la clave primaria del conjunto de relaciones.
- Un conjunto de entidades que no tiene suficientes atributos para formar una clave primaria se denomina **conjunto de entidades débiles**. Un conjunto de entidades que tiene una clave primaria se denomina **conjunto de entidades fuertes**.
- La **especialización** y la **generalización** definen una relación de contenido entre un conjunto de entidades de nivel más alto y uno o más conjuntos de entidades de nivel más bajo. La especialización es el resultado de tomar un subconjunto de un conjunto de entidades de nivel más alto para formar un conjunto de entidades de nivel más bajo. La generalización es el resultado de tomar la unión de dos o más conjuntos disjuntos de entidades (de nivel más bajo) para producir un conjunto de entidades de nivel más alto. Los atributos de los conjuntos de entidades de nivel más alto los heredan los conjuntos de entidades de nivel más bajo.
- La **agregación** es una abstracción en la que los conjuntos de relaciones (junto con sus conjuntos de entidades asociados) se tratan como conjuntos de entidades de nivel más alto, y pueden participar en las relaciones.
- Las diferentes características del modelo E-R ofrecen al diseñador de bases de datos numerosas decisiones de cómo representar mejor la empresa que se modela. Los conceptos y objetos pueden, en ciertos casos, representarse mediante entidades, relaciones o atributos. Ciertos aspectos de la estructura global de la empresa se pueden describir mejor usando conjuntos de entidades débiles, generalización, especialización o agregación. A menudo el diseñador debe sopesar las ventajas de un modelo simple y compacto frente a otros más precisos pero más completos.
- Una base de datos que se representa en un diagrama E-R se puede representar mediante una colección de tablas. Para cada conjunto de entidades y para cada conjunto de relaciones de la base de datos hay una única tabla a la que se le asigna el nombre del conjunto de entidades o del conjunto de relaciones correspondiente. Cada tabla tiene un número de columnas, cada una de las cuales tiene un nombre único. La conversión de una representación de base de datos en un diagrama E-R a un formato de tabla se basa en la deri-

vación de un diseño de bases de datos relacional desde un diagrama E-R.

- El **lenguaje de modelado unificado (UML)** proporciona un medio gráfico de modelar varios componentes de un sistema software. El componente diagrama de clase de UML se basa en diagramas E-R. Sin embargo, hay algunas diferencias entre ambos que se deben tener presentes.

TÉRMINOS DE REPASO

- Agregación
- Atributo derivado
- Atributos
- Atributos descriptivos
- Atributos monovalorados y multivalorados
- Atributos simples y compuestos
- Conjunto de entidades
- Conjunto de relaciones
- Conjunto de relaciones binario
- Conjunto de relaciones recursivo
- Conjuntos de entidades débiles y fuertes
  - Atributos discriminantes
  - Relaciones identificadoras
- Correspondencia de cardinalidad:
  - Relación uno a uno
  - Relación uno a varios
  - Relación varios a uno
  - Relación varios a varios
- Diagrama E-R
- Dominio
- Entidad
- Especialización y generalización
  - Superclase y subclase
  - Herencia de atributos
  - Herencia simple y múltiple
  - Pertenencia definida por condición y definida por el usuario
  - Generalización disjunta y solapada
- Grado de un conjunto de relaciones
- Lenguaje de modelado unificado (UML)
- Modelo de datos entidad-relación
- Papel
- Participación
  - Participación total
  - Participación parcial
- Relación
- Restricción de completitud
  - Generalización total y parcial
- Superclave, clave candidata y clave primaria
- Valor nulo

EJERCICIOS

2.1. Explíquense las diferencias entre los términos clave primaria, clave candidata y superclave.

2.2. Constrúyase un diagrama E-R para una compañía de seguros de coches cuyos clientes poseen uno o más coches. Cada coche tiene asociado un número de cero a cualquier valor que almacena el número de accidentes.

2.3. Constrúyase un diagrama E-R para un hospital con un conjunto de pacientes y un conjunto de médicos. Asóciese con cada paciente un registro de las diferentes pruebas y exámenes realizados.

2.4. Una oficina de registro de una universidad mantiene datos acerca de las siguientes entidades: (a) asignaturas, incluyendo el número, título, programa, y prerrequisitos; (b) ofertas de asignaturas, incluyendo número de asignatura, año, semestre, número de sección, profesor(es), horarios y aulas; (c) estudiantes, incluyendo id-estudiante, nombre y programa; y (d) profesores, incluyendo número de identificación, nombre, departamento y título. Además, la matrícula de los estudiantes en asignaturas y las notas concedidas a estudiantes en cada asignatura en la que están matriculados se deben modelar adecuadamente.

Constrúyase un diagrama E-R para la oficina de registro. Documéntense todas las decisiones que se hagan acerca de restricciones de correspondencia.

2.5. Considérese una base de datos usada para registrar las notas que obtienen los estudiantes en diferentes exámenes de diferentes ofertas de asignaturas.

a. Constrúyase un diagrama E-R que modele exámenes como entidades y use una relación ternaria para esta base de datos.

b. Constrúyase un diagrama E-R alternativo que use sólo una relación binaria entre *estudiantes* y *ofertas-asignaturas*. Asegúrese de que sólo existe una relación entre un par determinado estudiante y oferta-asignatura y de que aún se pueden representar las notas que obtiene un estudiante en diferentes exámenes de una oferta de una asignatura.

2.6. Constrúyanse tablas apropiadas para cada uno de los diagramas E-R de los Ejercicios 2.2 al 2.4.

2.7. Diséñese un diagrama E-R para almacenar los logros de su equipo deportivo favorito. Se deberían almacenar los partidos jugados, los resultados de cada partido, los jugadores de cada partido y las estadísticas individuales de cada jugador para cada partido. Las estadísticas de resumen se deberían modelar como atributos derivados.

2.8. Extiéndase el diagrama E-R del ejercicio anterior para almacenar la misma información para todos los equipos de una liga.

2.9. Explíquense las diferencias entre conjunto de entidades débiles y fuertes.

2.10. Se puede convertir cualquier conjunto de entidades débiles en un conjunto de entidades fuertes simplemente añadiendo los atributos apropiados. ¿Por qué, entonces, se tienen conjuntos de entidades débiles?

2.11. Defínase el concepto de agregación. Propónganse ejemplos para los que este concepto es útil.

2.12. Considérese el diagrama de la Figura 2.29, que modela una librería en línea.

a. Lístense los conjuntos de entidades y sus claves primarias.

b. Supóngase que la librería añade casetes de música y discos compactos a su colección. El mismo elemento musical puede estar presente en formato de casete o de disco compacto con diferentes precios. Extiéndase el diagrama E-R para modelar esta adición, ignorando el efecto sobre las cestas de la compra.

c. Extiéndase ahora el diagrama E-R usando generalización para modelar el caso en que una cesta de la compra pueda contener cualquier combinación de libros, casetes de música o discos compactos.

2.13. Considérese un diagrama E-R en el que el mismo conjunto de entidades aparece varias veces. ¿Por qué está permitida esta redundancia, una mala práctica que se debería evitar siempre que sea posible?

2.14. Considérese una base de datos de una universidad para la planificación de las aulas para los exámenes finales. Esta base de datos se modelaría mediante un único conjunto de entidades *examen*, con atributos *nombre-asignatura*, *número-sección*, *número-aula* y *hora*. Alternativamente se podrían definir uno o más conjuntos de entidades, con conjuntos de relaciones para sustituir algunos de los atributos del conjunto de entidades *examen*, como

- *asignatura* con atributos *nombre, departamento* y *número-a*
- *sección* con atributos *número-s* y *matriculados*, que es un conjunto de entidades débiles dependiente de *curso*.
- *aula* con atributos *número-a, capacidad* y *edificio*.

a. Muéstrese en un diagrama E-R el uso de los tres conjuntos de entidades adicionales listados.

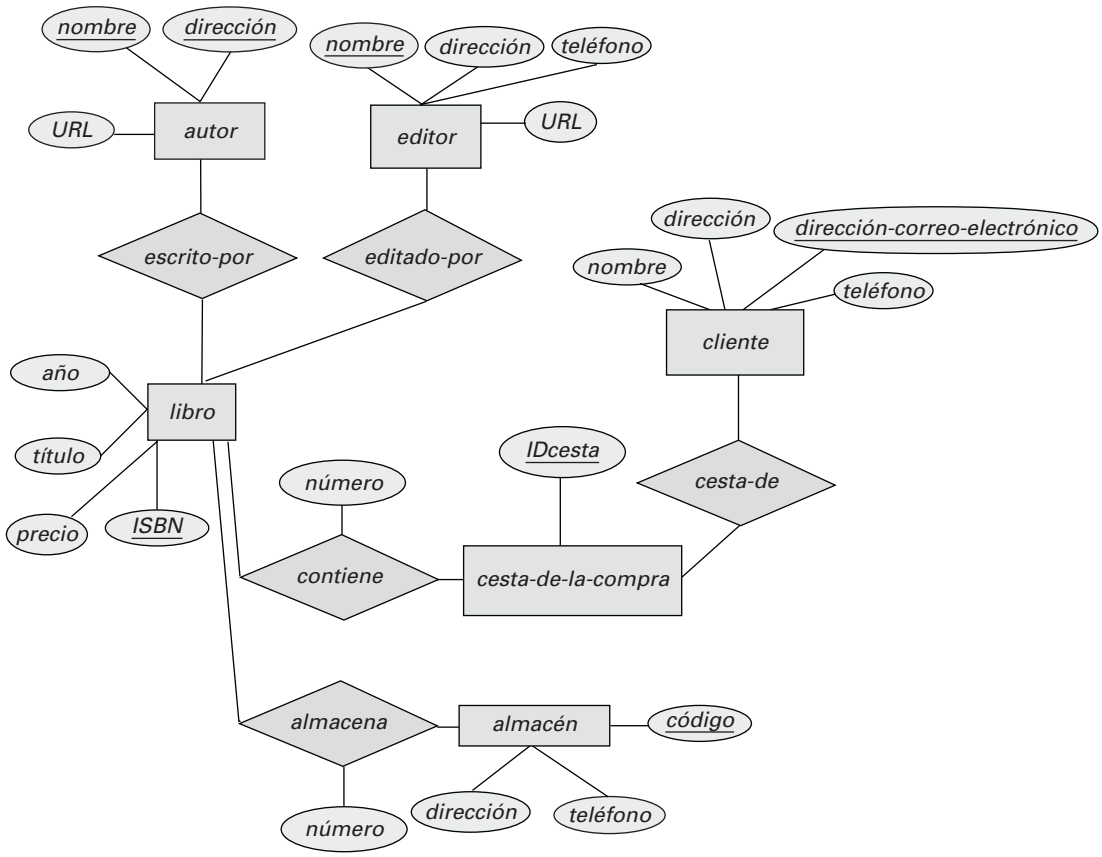

FIGURA 2.29. Diagrama E-R para el Ejercicio 2.12.

b. Explíquense las características que influirían en la decisión de incluir o no incluir cada uno de los conjuntos de entidades adicionales.

2.15. Cuando se diseña un diagrama E-R para un desarrollo particular se tienen varias alternativas entre las que hay que decidir.

a. ¿Qué criterio se deberá considerar para hacer la elección apropiada?

b. Diséñense tres alternativas de diagrama E-R para representar la oficina de registro de la universidad del Ejercicio 2.4. Lístense las ventajas de cada uno. Decídase por una de las alternativas.

2.16. Un diagrama E-R se puede ver como un grafo. ¿Qué significan los siguientes términos de estructura en un esquema de desarrollo?

a. El grafo es inconexo.

b. El grafo es acíclico.

2.17. En el Apartado 2.4.3 se representó una relación ternaria (Figura 2.30a) usando relaciones binarias, como se muestra en la Figura 2.30b. Considérese la alternativa mostrada en la Figura 2.30c. Discútanse las ventajas relativas a estas dos representaciones alternativas entre una relación ternaria y relaciones binarias.

2.18. Considérese la representación de una relación ternaria usando relaciones binarias como se describió en el Apartado 2.4.3 (mostrado en la figura 2.30b).

a. Muéstrese un ejemplar simple de E, A, B, C, R_A, R_B y R_C que no puedan corresponder a ningún ejemplar de A, B, C y R.

b. Modifíquese el diagrama E-R de la Figura 2.30b para introducir restricciones que garanticen que cualquier ejemplar E, A, B, C, R_A, R_B y R_C que satisfaga las restricciones corresponda a un ejemplar de A, B, C y R.

c. Modifíquese la traducción de arriba para manejar restricciones de participación total sobre las relaciones ternarias.

d. La representación de arriba requiere que se cree un atributo clave primaria para E. Muéstrese cómo tratar E como un conjunto de entidades débiles de forma que no se requiera un atributo clave primaria.

2.19. Un conjunto de entidades débiles siempre se puede convertir en un conjunto de entidades fuertes añadiéndole a sus atributos los atributos clave primaria de su conjunto de entidades identificadoras. Descríbase qué tipo de redundancia resultaría si se hiciese así.

2.20. Diséñese una jerarquía de especialización-generalización para las ventas de una compañía de vehículos a motor. La compañía vende motocicletas, coches de pasajeros, furgonetas y autobuses. Justifíquese la colocación de los atributos en cada nivel de la jerarquía. Explíquese por qué se deberían colocar en un nivel más alto o más bajo.

2.21. Explíquese la distinción entre las restricciones de diseño definidas por condición y las definidas por el usuario. ¿Cuáles de estas restricciones se pueden comprobar automáticamente? Explíquese la respuesta.

2.22. Explíquese la distinción entre las restricciones disjuntas y solapadas.

2.23. Explíquese la distinción entre las restricciones totales y parciales.

2.24. En la Figura 2.31 se muestra una estructura reticular de generalización y especialización. Para los conjuntos de entidades A, B y C explíquese cómo se heredan los atributos desde los conjuntos de entidades de nivel más alto X e Y. Discútase cómo manejar el caso en que un atributo de X tiene el mismo nombre que un atributo de Y.

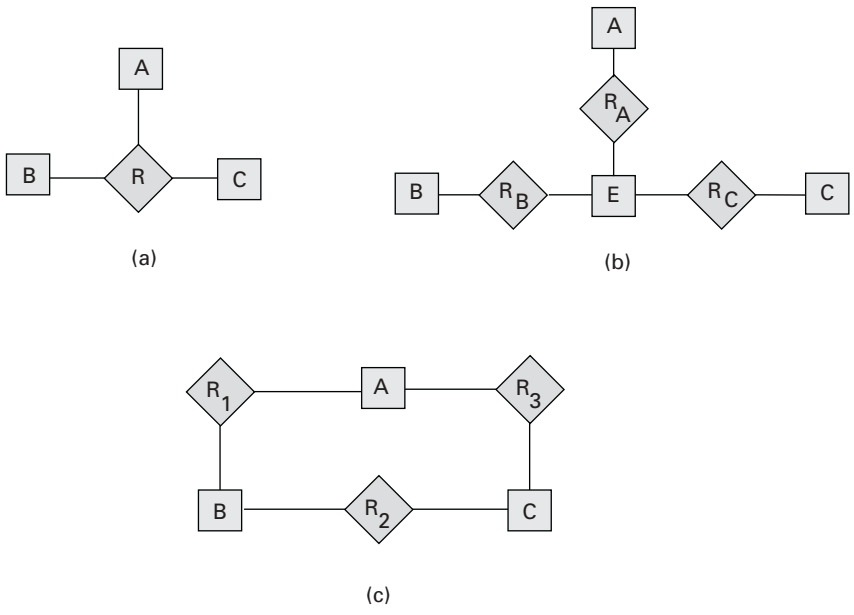

FIGURA 2.30. Diagrama E-R para el Ejercicio 2.17 (no se muestran los atributos).

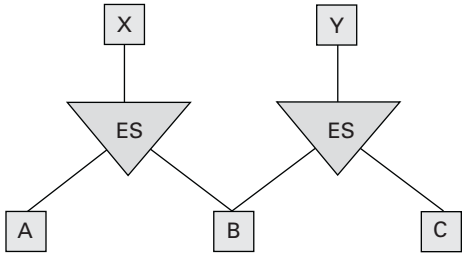

FIGURA 2.31. Diagrama E-R para el Ejercicio 2.18 (no se muestran los atributos).

2.25. Dibújense equivalentes UML de los diagramas E-R de las Figuras 2.9c, 2.10, 2.12, 2.13 y 2.17.

2.26. Considérense dos bancos separados que deciden fusionarse. Asúmase que ambos bancos usan exactamente el mismo esquema de bases de datos E-R, el de la Figura 2.22. (Obviamente, esta suposición es muy irreal; se considera un caso más realista en el Apartado 19.8.) Si la fusión del banco tiene una única base de datos, hay varios problemas potenciales:

- La posibilidad de que los dos bancos originales tengan sucursales con el mismo nombre.
- La posibilidad de que algunos clientes sean clientes de ambos bancos originales.
- La posibilidad de que algunos números de préstamo o de cuenta fueran usados en ambos bancos originales (para diferentes préstamos o cuentas, por supuesto).

Para cada uno de estos problemas potenciales descríbase por qué existen de hecho dificultades potenciales. Propóngase una solución a este problema. Explíquese cualquier cambio que se tendría que hacer para la solución y descríbase cómo afecta al esquema y a los datos.

2.27. Reconsidérese la situación descrita en el Ejercicio 2.26 bajo la suposición de que un banco está en España y el otro en Portugal. Por lo tanto, los bancos usan el esquema de la Figura 2.22, excepto que el banco portugués usa un número de identificación asignado por el gobierno portugués, mientras que el banco español usa el D.N.I. español para la identificación de clientes. ¿Qué problemas (además de los identificados en el Ejercicio 2.24) ocurrirían en este caso multinacional? ¿Cómo se podrían resolver? Asegúrese de considerar ambos esquemas y los valores de los datos actuales en la construcción de la respuesta.

NOTAS BIBLIOGRÁFICAS

El modelo de datos E-R fue introducido por Chen [1976]. Teorey et al. [1986] usó una metodología de diseño lógico para bases de datos relacionales que usa el modelo E-R extendido. La correspondencia entre los modelos E-R extendido y relacional se discutió por Lyngbaek y Vianu [1987], y por Marlowitz y Shoshani [1992]. Se han propuesto varios lenguajes de manipulación de datos para el modelo E-R: GERM (Benneworth et al. [1981]), GORDAS (Elmasri y Wiederhold [1981]) y ERROL (Markowitz y Raz [1983]). Un lenguaje de consulta gráfico para las bases de datos E-R se propuso por Zhang y Mendelzon [1983], y por ElMasri y Larson [1985].

Los conceptos de generalización, especialización y agregación se introdujeron por Smith y Smith [1977], y se extendieron en Hammer y McLeod [1981]. Lenzerini y Santucci [1983] han usado estos conceptos para definir las restricciones del modelo E-R.

Thalheim [2000] proporciona un tratamiento detallado de libros de texto de la investigación en el modelado E-R. En Batini et al. [1992] y Elmasri y Navathe [2000] se ofrecen discusiones sobre libros de texto básicos. Davis et al. [1983] proporciona una colección de artículos del modelo E-R.

HERRAMIENTAS

Muchos sistemas de bases de datos proporcionan herramientas para el diseño de bases de datos que soportan diagramas E-R. Estas herramientas ayudan al diseñador a crear diagramas E-R y pueden crear automáticamente las tablas correspondientes en una base de datos. Véanse las notas bibliográficas del Capítulo 1 para consultar referencias a sitios Web de fabricantes de bases de datos. Hay también algunas herramientas de modelado de datos independientes que soportan diagramas E-R y diagramas de clase UML. Entre ellas están Rational Rose (www.rational.com/products/rose). Visio Enterprise (véase www.visio.com) y ERwin (búsquese ERwin en el sitio www.cai.com/products).

CAPÍTULO

3 EL MODELO RELACIONAL

El modelo relacional se ha establecido actualmente como el principal modelo de datos para las aplicaciones de procesamiento de datos. Ha conseguido la posición principal debido a su simplicidad, que facilita el trabajo del programador en comparación con otros modelos anteriores como el de red y el jerárquico.

En este capítulo se estudia en primer lugar los fundamentos del modelo relacional, que proporciona una forma muy simple y potente de representar datos. A continuación se describen tres lenguajes formales de consulta; los lenguajes de consulta se usan para especificar las solicitudes de información. Los tres que se estudian en este capítulo no son cómodos de usar, pero a cambio sirven como base formal para lenguajes de consulta que sí lo son y que se estudiarán más adelante. El primer lenguaje de consulta, el álgebra relacional, se estudia en detalle. El álgebra relacional forma la base del lenguaje de consulta SQL ampliamente usado. A continuación se proporcionan visiones generales de otros dos lenguajes formales: el cálculo relacional de tuplas y el cálculo relacional de dominios, que son lenguajes declarativos de consulta basados en la lógica matemática. El cálculo relacional de dominios es la base del lenguaje QBE.

Existe una amplia base teórica de las bases de datos relacionales. En este capítulo se estudia la base teórica referida a las consultas. En el Capítulo 7 se examinarán aspectos de la teoría de bases de datos relacionales que ayudan en el diseño de esquemas de bases de datos relacionales, mientras que en los Capítulos 13 y 14 se estudian aspectos de la teoría que se refieren al procesamiento eficiente de consultas.

3.1. LA ESTRUCTURA DE LAS BASES DE DATOS RELACIONALES

Una base de datos relacional consiste en un conjunto de **tablas**, a cada una de las cuales se le asigna un nombre exclusivo. Cada tabla tiene una estructura parecida a la presentada en el Capítulo 2, donde se representaron las bases de datos E-R mediante tablas. Cada fila de la tabla representa una *relación* entre un conjunto de valores. Dado que cada tabla es un conjunto de dichas relaciones, hay una fuerte correspondencia entre el concepto de *tabla* y el concepto matemático de *relación*, del que toma su nombre el modelo de datos relacional. A continuación se introduce el concepto de relación.

En este capítulo se utilizarán varias relaciones diferentes para ilustrar los conceptos subyacentes al modelo de datos relacional. Estas relaciones representan parte de una entidad bancaria. Se diferencian ligeramente de las tablas que se utilizaron en el Capítulo 2, por lo que se puede simplificar la representación. En el Capítulo 7 se estudiarán los criterios sobre la adecuación de las estructuras relacionales.

3.1.1. Estructura básica

Considérese la tabla *cuenta* de la Figura 3.1. Tiene tres cabeceras de columna: *número-cuenta*, *nombre-sucursal* y *saldo*. Siguiendo la terminología del modelo relacional se puede hacer referencia a estas cabeceras como **atributos** (igual que se hizo en el modelo E-R en el Capítulo 2). Para cada atributo hay un conjunto de valores permitidos, llamado **dominio** de ese atributo. Para el atributo *nombre-sucursal*, por ejemplo, el dominio es el conjunto de los nombres de las sucursales. Supongamos que D_1 denota el conjunto de todos los números de cuenta, D_2 el conjunto de todos los nombres de sucursal y D_3 el conjunto de los saldos. Como se vio en el Capítulo 2 todas las filas de *cuenta* deben consistir en una tupla (v_1, v_2, v_3), donde v_1 es un número de cuenta (es decir, v_1 está en el dominio D_1), v_2 es un nombre de sucursal (es decir, v_2 está en el dominio D_2) y v_3 es un saldo (es decir, v_3 está en el dominio D_3). En general,

número-cuenta	*nombre-sucursal*	*saldo*
C-101	Centro	500
C-102	Navacerrada	400
C-201	Galapagar	900
C-215	Becerril	700
C-217	Galapagar	750
C-222	Moralzarzal	700
C-305	Collado Mediano	350

FIGURA 3.1. La relación *cuenta*.

cuenta sólo contendrá un subconjunto del conjunto de todas las filas posibles. Por tanto, *cuenta* es un subconjunto de

$$D_1 \times D_2 \times D_3$$

En general, una **tabla** de n atributos debe ser un subconjunto de

$$D_1 \times D_2 \times \ldots \times D_{n-1} \times D_n$$

Los matemáticos definen las **relaciones** como subconjuntos del producto cartesiano de la lista de dominios. Esta definición se corresponde de manera casi exacta con la definición de tabla dada anteriormente. La única diferencia es que aquí se han asignado nombres a los atributos, mientras que los matemáticos sólo utilizan «nombres» numéricos, utilizando el entero 1 para denotar el atributo cuyo dominio aparece en primer lugar en la lista de dominios, 2 para el atributo cuyo dominio aparece en segundo lugar, etcétera. Como las tablas son esencialmente relaciones, se utilizarán los términos matemáticos **relación** y **tupla** en lugar de los términos **tabla** y **fila**. Una **variable tupla** es una variable que representa a una tupla; en otras palabras, una tupla que representa al conjunto de todas las tuplas.

En la relación *cuenta* de la Figura 3.1 hay siete tuplas. Supóngase que la variable tupla t hace referencia a la primera tupla de la relación. Se utiliza la notación t[*número-cuenta*] para denotar el valor de t en el atributo *número-cuenta*. Por tanto, t[*número-cuenta*] = «C-101» y t[*nombre-sucursal*] = «Centro». De manera alternativa, se puede escribir t[1] para denotar el valor de la tupla t en el primer atributo (*número-cuenta*), t[2] para denotar *nombre-sucursal*, etcétera. Dado que las relaciones son conjuntos se utiliza la notación matemática $t \in r$ para denotar que la tupla t está en la relación r.

El orden en que aparecen las tuplas es irrelevante, dado que una relación es un *conjunto* de tuplas. Así, si las tuplas de una relación se muestran ordenadas como en la Figura 3.1, o desordenadas, como en la Figura 3.2, no importa; las relaciones de estas figuras son las mismas, ya que ambas contienen el mismo conjunto de tuplas.

Se exigirá que, para todas las relaciones r, los dominios de todos los atributos de r sean atómicos. Un dominio es **atómico** si los elementos del dominio se consideran unidades indivisibles. Por ejemplo, el conjunto de los enteros es un dominio atómico, pero el conjunto de todos los conjuntos de enteros es un dominio no atómico. La diferencia es que no se suele considerar que los enteros tengan subpartes, pero sí se considera que los conjuntos de enteros las tienen; por ejemplo, los enteros que forman cada conjunto. Lo importante no es lo que sea el propio dominio, sino la manera en que se utilizan los elementos del dominio en la base de datos. El dominio de todos los enteros sería no atómico si se considerase que cada entero fuera una lista ordenada de cifras. En todos los ejemplos se supondrá que los dominios son atómicos. En el Capítulo 9 se estudiarán extensiones al modelo de datos relacional para permitir dominios no atómicos.

número-cuenta	*nombre-sucursal*	*saldo*
C-101	Centro	500
C-215	Becerril	700
C-102	Navacerrada	400
C-305	Collado Mediano	350
C-201	Galapagar	900
C-222	Moralzarzal	700
C-217	Galapagar	750

FIGURA 3.2. La relación *cuenta* con las tuplas desordenadas.

Es posible que varios atributos tengan el mismo dominio. Por ejemplo, supóngase que se tiene una relación *cliente* que tiene los tres atributos *nombre-cliente*, *calle-cliente* y *ciudad-cliente* y una relación *empleado* que incluye el atributo *nombre-empleado*. Es posible que los atributos *nombre-cliente* y *nombre-empleado* tengan el mismo dominio, el conjunto de todos los nombres de personas, que en el nivel físico son cadenas de caracteres. Los dominios de *saldo* y *nombre-sucursal*, por otra parte, deberían ser distintos. Quizás es menos claro si *nombre-cliente* y *nombre-sucursal* deberían tener el mismo dominio. En el nivel físico, tanto los nombres de clientes como los nombres de sucursales son cadenas de caracteres. Sin embargo, en el nivel lógico puede que se desee que *nombre-cliente* y *nombre-sucursal* tengan dominios diferentes.

Un valor de dominio que es miembro de todos los dominios posibles es el valor **nulo**, que indica que el valor es desconocido o no existe. Por ejemplo, supóngase que se incluye el atributo *número-teléfono* en la relación *cliente*. Puede ocurrir que un cliente no tenga número de teléfono, o que su número de teléfono no figure en la guía. Entonces habrá que recurrir a los valores nulos para indicar que el valor es desconocido o que no existe. Más adelante se verá que los valores nulos crean algunas dificultades cuando se tiene acceso a la base de datos o cuando se actualiza y que, por tanto, deben eliminarse si es posible. Se asumirá inicialmente que no hay valores nulos y en el Apartado 3.3.4 se describirá el efecto de los valores nulos en las diferentes operaciones.

3.1.2. Esquema de la base de datos

Cuando se habla de bases de datos se debe diferenciar entre el **esquema de la base de datos**, o diseño lógico de la misma, y el **ejemplar de la base de datos**, que es una instantánea de los datos de la misma en un momento dado.

El concepto de relación se corresponde con el concepto de variable de los lenguajes de programación. El concepto de **esquema de la relación** se corresponde con el concepto de definición de tipos de los lenguajes de programación.

Resulta conveniente dar un nombre a los esquemas de las relaciones, igual que se dan nombres a las definiciones de tipos en los lenguajes de programación. Se adopta el convenio de utilizar nombres en minúsculas para las relaciones y nombres que comiencen por una letra mayúscula para los esquemas de las relaciones. Siguiendo esta notación se utilizará *Esquema-cuenta* para denotar el esquema de la relación de la relación *cuenta*. Por tanto,

Esquema-cuenta = (*número-cuenta, nombre-sucursal, saldo*)

Se denota el hecho de que *cuenta* es una relación de *Esquema-cuenta* mediante

cuenta (*Esquema-cuenta*)

En general, los esquemas de las relaciones incluyen una lista de los atributos y de sus dominios correspondientes. La definición exacta del dominio de cada atributo no será relevante hasta que se discuta el lenguaje SQL en el Capítulo 4.

El concepto de **ejemplar de relación** se corresponde con el concepto de valor de una variable en los lenguajes de programación. El valor de una variable dada puede cambiar con el tiempo; de manera parecida, el contenido del ejemplar de una relación puede cambiar con el tiempo cuando la relación se actualiza. Sin embargo, se suele decir simplemente «relación» cuando realmente se quiere decir «ejemplar de la relación».

Como ejemplo de ejemplar de una relación, considérese la relación *sucursal* de la Figura 3.3. El esquema de esa relación es

Esquema-relación = (*nombre-sucursal, ciudad-sucursal, activos*)

Obsérvese que el atributo *nombre de la sucursal* aparece tanto en *Esquema-sucursal* como en *Esquema-cuenta*. Esta duplicidad no es una coincidencia. Más bien, utilizar atributos comunes en los esquemas de las relaciones es una manera de relacionar las tuplas de relaciones diferentes. Por ejemplo, supóngase que se desea obtener información sobre todas las cuentas abiertas en sucursales ubicadas en Arganzuela. Primero se busca en la relación *sucursal* para encontrar los nombres de todas las sucursales sitas en Arganzuela. Luego, para cada una de ellas, se mira en la relación *cuenta* para encontrar la información sobre las cuentas abiertas en esa sucursal. Esto no es sorprendente: recuérdese que los atributos que forma la clave primaria de un conjunto de entidades fuertes aparecen en la tabla creada para representar el conjunto de entidades, así como en las tablas creadas para crear relaciones en las que participar el conjunto de entidades.

Continuemos con el ejemplo bancario. Se necesita una relación que describa la información sobre los clientes. El esquema de la relación es:

Esquema-cliente = (*nombre-cliente, calle-cliente, ciudad-cliente*)

En la Figura 3.4 se muestra un ejemplo de la relación *cliente* (*Esquema-cliente*). Obsérvese que se ha omitido el atributo *id-cliente*, que se usó en el Capítulo 2, porque no se desea tener esquemas de relación más pequeños en este ejemplo. Se asume que el nombre de cliente identifica unívocamente un cliente; obviamente, esto no es cierto en el mundo real, pero las suposiciones hechas en estos ejemplos los hacen más sencillos de entender.

En una base de datos del mundo real, *id-cliente* (que podría ser el número de la seguridad social o un identificador generado por el banco) serviría para identificar unívocamente a los clientes.

También se necesita una relación que describa la asociación entre los clientes y las cuentas. El esquema de la relación que describe esta asociación es:

Esquema-impositor = (*nombre-cliente, número-cuenta*)

En la Figura 3.5 se muestra un ejemplo de la relación *impositor* (*Esquema-impositor*).

Puede parecer que, para el presente ejemplo bancario, se podría tener sólo un esquema de relación, en vez de tener varios. Es decir, puede resultar más sencillo para el usuario pensar en términos de un esquema de

nombre de la sucursal	*ciudad de la sucursal*	*activos*
Galapagar	Arganzuela	7.500
Centro	Arganzuela	9.000.000
Becerril	Aluche	2.000
Segovia	Cerceda	3.700.000
Navacerrada	Aluche	1.700.000
Navas de la Asunción	Alcalá de Henares	1.500
Moralzarzal	La Granja	2.500
Collado Mediano	Aluche	8.000.000

FIGURA 3.3. La relación *sucursal*.

nombre-cliente	*calle-cliente*	*ciudad-cliente*
Abril	Preciados	Valsaín
Amo	Embajadores	Arganzuela
Badorrey	Delicias	Valsaín
Fernández	Jazmín	León
Gómez	Carretas	Cerceda
González	Arenal	La Granja
López	Mayor	Peguerinos
Pérez	Carretas	Cerceda
Rodríguez	Yeserías	Cádiz
Rupérez	Ramblas	León
Santos	Mayor	Peguerinos
Valdivieso	Goya	Vigo

FIGURA 3.4. La relación *cliente*.

nombre cliente	*número cuenta*
Abril	C-102
Gómez	C-101
González	C-201
González	C-217
López	C-222
Rupérez	C-215
Santos	C-305

FIGURA 3.5. La relación *impositor*.

número-préstamo	*nombre-sucursal*	*importe*
P-11	Collado Mediano	900
P-14	Centro	1.500
P-15	Navacerrada	1.500
P-16	Navacerrada	1.300
P-17	Centro	1.000
P-23	Moralzarzal	2.000
P-93	Becerril	500

FIGURA 3.6. La relación *préstamo*.

relación, en lugar de en términos de varios esquemas. Supóngase que sólo se utilizara una relación para el ejemplo, con el esquema

(*nombre-sucursal, ciudad-sucursal, activos, nombre-cliente, calle-cliente, ciudad-cliente, número-cuenta, saldo*)

Obsérvese que si un cliente tiene varias cuentas hay que repetir su dirección una vez por cada cuenta. Es decir, hay que repetir varias veces parte de la información. Esta repetición supone un gasto inútil y se evita mediante el uso de varias relaciones, como en el ejemplo presente.

Además, si una sucursal no tiene ninguna cuenta (por ejemplo, una sucursal recién creada que todavía no tiene clientes), no se puede construir una tupla completa en la relación única anterior, dado que no hay todavía ningún dato disponible referente a *cliente* ni a *cuenta*. Para representar las tuplas incompletas hay que utilizar valores *nulos* que indiquen que ese valor es desconocido o no existe. Por tanto, en el ejemplo presente, los valores de *nombre-cliente*, *calle-cliente*, etcétera, deben quedar nulos. Utilizando varias relaciones se puede representar la información de las sucursales de un banco sin clientes sin utilizar valores nulos. Sencillamente, se utiliza una tupla en *Esquema-sucursal* para representar la información de la sucursal y sólo crear tuplas en los otros esquemas cuando esté disponible la información adecuada.

En el Capítulo 7 se estudiarán los criterios para decidir cuándo un conjunto de esquemas de relaciones es más apropiado que otro en términos de repetición de la información y de la existencia de valores nulos. Por ahora se supondrá que los esquemas de las relaciones vienen dados de antemano.

Se incluyen dos relaciones más para describir los datos de los préstamos concedidos en las diferentes sucursales del banco:

Esquema-préstamo = (*número-préstamo, nombre-sucursal, importe*)
Esquema-prestatario = (*nombre-cliente, número-préstamo*)

Las relaciones de ejemplo *préstamo* (*Esquema-préstamo*) y *prestatario* (*Esquema-prestatario*) se muestran en las Figuras 3.5 y 3.6, respectivamente.

La entidad bancaria que se ha descrito se deriva del diagrama E-R mostrado en la Figura 3.8. Los esquemas de las relaciones se corresponden con el conjunto de tablas que se podrían generar utilizando el método esbozado en el Apartado 2.9. Obsérvese que las tablas para *cuenta-sucursal* y *préstamo-sucursal* se han combinado en las tablas de *cuenta* y *préstamo* respectivamente. Esta combinación es posible dado que las relaciones son de varios a uno desde *cuenta* y *préstamo*, respectivamente, a *sucursal* y, además, la participación de *cuenta* y *préstamo* en las relaciones correspondientes es total, como indican las líneas dobles en la figura. Finalmente, obsérvese que la relación *cliente* puede contener información sobre clientes que ni tengan cuenta ni un préstamo en el banco.

La entidad bancaria aquí descrita servirá como ejemplo principal en este capítulo y en los siguientes. Cuando sea necesario, habrá que introducir más esquemas de relaciones para ilustrar casos concretos.

3.1.3. Claves

Los conceptos de *superclave*, de *clave candidata* y de *clave primaria*, tal y como se discute en el Capítulo 2, también son aplicables en el modelo relacional. Por ejemplo, en *Esquema-sucursal*, tanto {*nombre-sucursal*} como {*nombre-sucursal*, *ciudad-sucursal*} son superclaves. {*nombre-sucursal*, *ciudad-sucursal*} no es una clave candidata porque {*nombre-sucursal*} es un subconjunto de {*nombre-sucursal*, *ciudad-sucursal*} y {*nombre-sucursal*} es una superclave. Sin embargo, {*nombre-sucursal*} es una clave candidata, y servirá también como clave primaria para estos fines. El atributo *ciudad-sucursal* no es una superclave, dado que dos sucursales de la misma ciudad pueden tener nombres diferentes (y diferentes volúmenes de activos).

nombre cliente	*número préstamo*
Fernández	P-16
Gómez	P-93
Gómez	P-15
López	P-14
Pérez	P-17
Santos	P-11
Sotoca	P-23
Valdivieso	P-17

FIGURA 3.7. La relación *prestatario*.

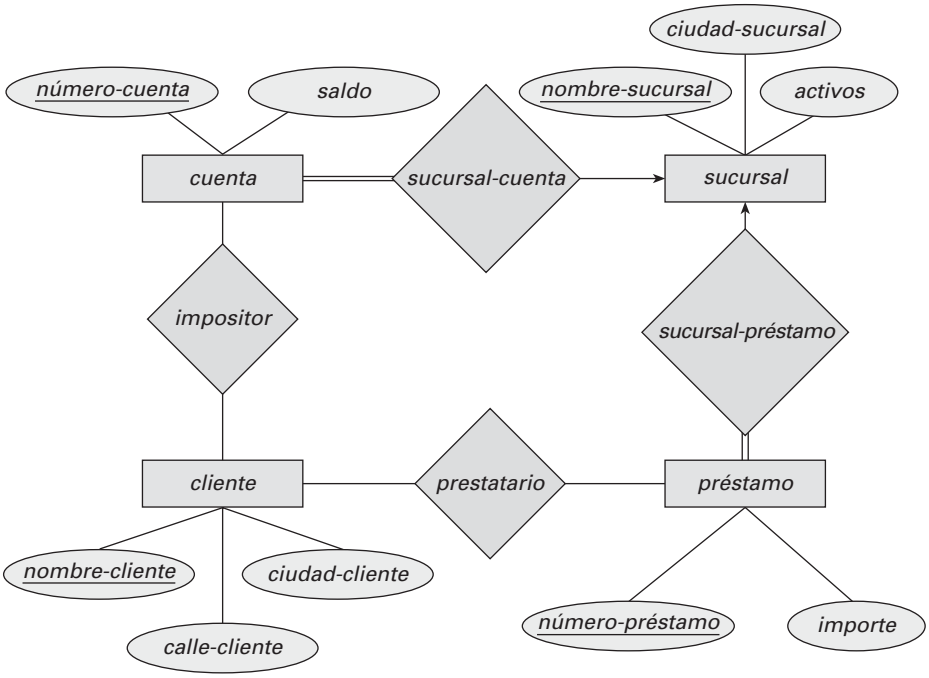

FIGURA 3.8. Diagrama E-R de la entidad bancaria.

Sea R el esquema de una relación. Si se dice que un subconjunto K de R es una *superclave* de R para las relaciones $r(R)$ en las que no hay dos tuplas diferentes que tengan los mismos valores en todos los atributos de K. Es decir, si t_1 y t_2 están en R y $t_1 \neq t_2$, entonces $t_1[K] \neq t_2[K]$.

Si el esquema de una base de datos relacional se basa en las tablas derivadas de un esquema E-R es posible determinar la clave primaria del esquema de una relación a partir de las claves primarias de los conjuntos de entidades o de relaciones de los que se deriva el esquema:

- **Conjunto de entidades fuertes.** La clave primaria del conjunto de entidades se convierte en la clave primaria de la relación.
- **Conjunto de entidades débiles.** La tabla y, por tanto, la relación correspondientes a un conjunto de entidades débiles incluyen
  - — Los atributos del conjunto de entidades débiles.
  - — La clave primaria del conjunto de entidades fuertes del que depende el conjunto de entidades débiles.

  La clave primaria de la relación consiste en la unión de la clave primaria del conjunto de entidades fuertes y el discriminante del conjunto de entidades débil.
- **Conjunto de relaciones.** La unión de las claves primarias de los conjuntos de entidades relacionados se transforma en una superclave de la relación. Si la relación es de varios a varios, esta superclave es también la clave primaria. En el Apartado 2.4.2 se describe la manera de determinar las claves primarias en otros casos. Recuérdese del Apartado 2.9.3 que no se genera ninguna tabla para los conjuntos de relaciones que vinculan un conjunto de entidades débiles con el conjunto de entidades fuertes correspondiente.
- **Tablas combinadas.** Recuérdese del Apartado 2.9.3 que un conjunto binario de relaciones de varios a uno entre A y B puede representarse mediante una tabla que consista en los atributos de A y en los atributos (si hay alguno) del conjunto de relaciones. La clave primaria de la entidad «varios» se transforma en la clave primaria de la relación (es decir, si el conjunto de relaciones es de varios a uno entre A y B, la clave primaria de A es la clave primaria de la relación). Para los conjuntos de relaciones de uno a uno la relación se construye igual que en el conjunto de relaciones de varios a uno. Sin embargo, cualquiera de las claves primarias del conjunto de entidades puede elegirse como clave primaria de la relación, dado que ambas son claves candidatas.
- **Atributos multivalorados.** Recuérdese del Apartado 2.9.4 que un atributo multivalorado M se representa mediante una tabla consistente en la clave primaria del conjunto de entidades o de relaciones del que M es atributo y en una columna C que guarda un valor concreto de M. La clave primaria del conjunto de entidades o de relaciones

junto con el atributo C se convierte en la clave primaria de la relación.

A partir de la lista precedente se puede ver que el esquema de una relación puede incluir entre sus atributos la clave primaria de otro esquema, digamos r_2. Este atributo es una **clave externa** de r_1 que hace referencia a r_2. La relación r_1 también se denomina la **relación referenciante** de la dependencia de clave externa, y r_2 se denomina la **relación referenciada** de la clave externa. Por ejemplo, el atributo *nombre-sucursal* de *Esquema-cuenta* es una clave externa de *Esquema-cuenta* referenciando a *Esquema-sucursal*, ya que *nombre-sucursal* es la clave primaria de *Esquema-sucursal*. En cualquier ejemplar de la base de datos, dada una tupla t_a de la relación *cuenta*, debe haber alguna tupla t_b en la relación *cuenta* tal que el valor del atributo *nombre-sucursal* de t_a sea el mismo que el valor de la clave primaria, *nombre-sucursal*, de t_b.

Es obligado listar los atributos que forman clave primaria de un esquema de relación antes que el resto; por ejemplo, el atributo *nombre-sucursal* de *Esquema-sucursal* se lista en primer lugar, ya que es la clave primaria.

3.1.4. Diagramas de esquema

Un esquema de bases de datos, junto con las dependencias de clave primaria y externa, se puede mostrar gráficamente mediante **diagramas de esquema**. La Figura 3.9 muestra el diagrama de esquema del ejemplo bancario. Cada relación aparece como un cuadro con los atributos listados dentro de él y el nombre de la relación sobre él. Si hay atributos clave primaria, una línea horizontal cruza el cuadro con los atributos clave primaria listados sobre ella. Las dependencias de clave externa aparecen como flechas desde los atributos clave externa de la relación referenciante a la clave primaria de la relación referenciada.

No hay que confundir un diagrama de esquema con un diagrama E-R. En particular, los diagramas E-R no muestran explícitamente los atributos clave externa, mientras que los diagramas de esquema sí.

Muchos sistemas de bases de datos proporcionan herramientas de diseño con una interfaz gráfica de usuario para la creación de diagramas de esquema.

3.1.5. Lenguajes de consulta

Un **lenguaje de consulta** es un lenguaje en el que un usuario solicita información de la base de datos. Estos lenguajes suelen ser de un nivel superior que el de los lenguajes de programación habituales. Los lenguajes de consulta pueden clasificarse como procedimentales o no procedimentales. En los **lenguajes procedimentales** el usuario instruye al sistema para que lleve a cabo una serie de operaciones en la base de datos para calcular el resultado deseado. En los **lenguajes no procedimentales** el usuario describe la información deseada sin dar un procedimiento concreto para obtener esa información.

La mayor parte de los sistemas comerciales de bases de datos relacionales ofrecen un lenguaje de consulta que incluye elementos de los enfoques procedimental y no procedimental. Se estudiarán varios lenguajes comerciales en el Capítulo 4. El Capítulo 5 trata los lenguajes QBE y Datalog, este último parecido a Prolog.

En este capítulo se examinarán los lenguajes «puros»: el álgebra relacional es procedimental, mientras que el cálculo relacional de tuplas y el de dominios son no procedimentales. Estos lenguajes de consulta son rígidos y formales, y carecen del «azúcar sintáctico» de los lenguajes comerciales, pero ilustran las técnicas fundamentales para la extracción de datos de las bases de datos.

Aunque inicialmente sólo se estudiarán las consultas, un lenguaje de manipulación de datos completo no sólo incluye un lenguaje de consulta, sino también un lenguaje para la modificación de las bases de datos. Estos lenguajes incluyen órdenes para insertar y borrar tuplas, así como órdenes para modificar partes de las tuplas existentes. Las modificaciones de las bases de datos se examinarán después de completar la discusión sobre las consultas.

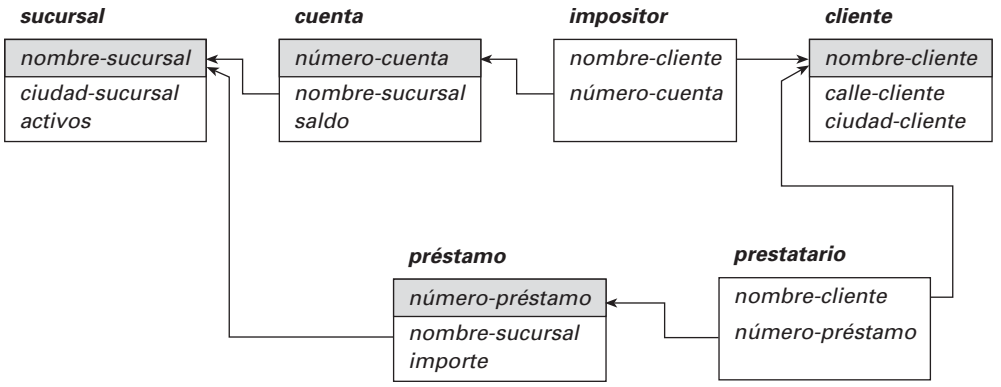

FIGURA 3.9. Diagrama de esquema para el banco.

3.2. EL ÁLGEBRA RELACIONAL

El álgebra relacional es un lenguaje de consulta *procedimental*. Consta de un conjunto de operaciones que toman como entrada una o dos relaciones y producen como resultado una nueva relación. Las operaciones fundamentales del álgebra relacional son *selección*, *proyección*, *unión*, *diferencia de conjuntos*, *producto cartesiano* y *renombramiento*. Además de las operaciones fundamentales hay otras operaciones, por ejemplo, intersección de conjuntos, reunión natural, división y asignación. Estas operaciones se definirán en términos de las operaciones fundamentales.

3.2.1. Operaciones fundamentales

Las operaciones selección, proyección y renombramiento se denominan operaciones *unarias* porque operan sobre una sola relación. Las otras tres operaciones operan sobre pares de relaciones y se denominan, por lo tanto, operaciones *binarias*.

3.2.1.1. La operación selección

La operación **selección** selecciona tuplas que satisfacen un predicado dado. Se utiliza la letra griega sigma minúscula (σ) para denotar la selección. El predicado aparece como subíndice de σ. La relación del argumento se da entre paréntesis a continuación de σ. Por tanto, para seleccionar las tuplas de la relación *préstamo* en que la sucursal es «Navacerrada» hay que escribir

$$\sigma_{nombre\text{-}sucursal = \text{«Navacerrada»}} (préstamo)$$

Si la relación *préstamo* es como se muestra en la Figura 3.6, la relación que resulta de la consulta anterior es como se muestra en la Figura 3.10.

Se pueden buscar todas las tuplas en las que el importe prestado sea mayor que 1.200 € escribiendo

$$\sigma_{importe>1200} (préstamo)$$

En general, se permiten las comparaciones que utilizan $=, \neq, <, \leq, >$ o $\geq$ en el predicado de selección. Además, se pueden combinar varios predicados en uno mayor utilizando las conectivas *y* ($\wedge$) y *o* ($\vee$). Por tanto, para encontrar las tuplas correspondientes a préstamos de más de 1.200 € concedidos por la sucursal de Navacerrada, se escribe

$$\sigma_{nombre\text{-}sucursal = \text{«Navacerrada»} \wedge importe>1200} (préstamo)$$

número-préstamo	*nombre-sucursal*	*importe*
P-15	Navacerrada	1.500
P-16	Navacerrada	1.300

FIGURA 3.10. Resultado de $\sigma_{\text{nombre-sucursal = «Navacerrada»}}$ (préstamo).

El predicado de selección puede incluir comparaciones entre dos atributos. Para ilustrarlo, considérese la relación *responsable-préstamo*, que consta de tres atributos: *nombre-cliente*, *nombre-banquero* y *número-préstamo*, que especifica que un empleado concreto es el responsable del préstamo concedido a un cliente. Para hallar todos los clientes que se llaman igual que su responsable de préstamos se puede escribir

$$\sigma_{nombre\text{-}cliente = nombre\text{-}banquero} (responsable\text{-}préstamo)$$

Dado que el valor especial *nulo* indica «valor desconocido o inexistente», cualquier comparación que implique a un valor nulo se evalúa como **falsa**.

3.2.1.2. La operación proyección

Supóngase que se desea hacer una lista de todos los números de préstamo y del importe de los mismos, pero sin que aparezcan los nombres de las sucursales. La operación **proyección** permite producir esta relación. La operación proyección es una operación unaria que devuelve su relación de argumentos, excluyendo algunos argumentos. Dado que las relaciones son conjuntos, se eliminan todas las filas duplicadas. La proyección se denota por la letra griega mayúscula pi (Π). Se crea una lista de los atributos que se desea que aparezcan en el resultado como subíndice de Π. La relación de argumentos se escribe a continuación entre paréntesis. Por tanto, la consulta para crear una lista de todos los números de préstamo y del importe de los mismos puede escribirse como

$$\Pi_{número\text{-}préstamo,\ importe} (préstamo)$$

La relación que resulta de esta consulta se muestra en la Figura 3.11.

3.2.1.3. Composición de operaciones relacionales

Es importante el hecho de que el resultado de una operación relacional sea también una relación. Considérese la consulta más compleja «Encontrar los clientes que viven en Peguerinos». Hay que escribir:

número-préstamo	*importe*
P-11	900
P-14	1.500
P-15	1.500
P-16	1.300
P-17	1.000
P-23	2.000
P-93	500

FIGURA 3.11. Números de préstamo y sus importes.

$$\Pi_{nombre\text{-}cliente}\,(\sigma_{ciudad\text{-}cliente\,=\,\text{«Peguerinos»}}\,(cliente))$$

Téngase en cuenta que, en vez de dar en el argumento de la operación proyección el nombre de una relación, se da una expresión que se evalúa como una relación.

En general, dado que el resultado de una operación del álgebra relacional es del mismo tipo (relación) que los datos de entrada, las operaciones del álgebra relacional pueden componerse para formar una **expresión del álgebra relacional**. La composición de operaciones del álgebra relacional para formar expresiones del álgebra relacional es igual que la composición de operaciones aritméticas (como +, –, ∗ y ÷) para formar expresiones aritméticas. La definición formal de las expresiones de álgebra relacional se estudia en el Apartado 3.2.2.

3.2.1.4. La operación unión

Considérese una consulta para averiguar el nombre de todos los clientes del banco que tienen una cuenta, un préstamo o ambas cosas. Obsérvese que la relación *cliente* no contiene esa información, dado que los clientes no necesitan tener ni cuenta ni préstamo en el banco. Para contestar a esta consulta hace falta la información de la relación *impositor* (Figura 3.5) y la de la relación *prestatario* (Figura 3.7). Se conoce la manera de averiguar los nombres de todos los clientes con préstamos en el banco:

$$\Pi_{nombre\text{-}cliente}\,(prestatario)$$

También se conoce la manera de averiguar el nombre de los clientes con cuenta en el banco:

$$\Pi_{nombre\text{-}cliente}\,(impositor)$$

Para contestar a la consulta hace falta la **unión** de estos dos conjuntos; es decir, hacen falta todos los nombres de clientes que aparecen en alguna de las dos relaciones o en ambas. Estos datos se pueden averiguar mediante la operación binaria unión, denotada, como en la teoría de conjuntos, por ∪. Por tanto, la expresión buscada es

$$\Pi_{nombre\text{-}cliente}\,(prestatario) \cup \Pi_{nombre\text{-}cliente}\,(impositor)$$

La relación resultante de esta consulta aparece en la Figura 3.10. Téngase en cuenta que en el resultado hay diez tuplas, aunque hay siete prestatarios y seis impositores distintos. Esta discrepancia aparente se debe a que Gómez, Santos y López son a la vez prestatarios e impositores. Dado que las relaciones son conjuntos, se eliminan los valores duplicados.

Obsérvese que en este ejemplo se toma la unión de dos conjuntos, ambos consistentes en valores de *nombre-cliente*. En general, se debe asegurar que las uniones se realicen entre relaciones *compatibles*. Por ejemplo, no tendría sentido realizar la unión de las relaciones *préstamo* y *prestatario*. La primera es una relación con tres atributos, la segunda sólo tiene dos. Más aún, considérese la unión de un conjunto de nombres de clientes y de un conjunto de ciudades. Una unión así no tendría sentido en la mayor parte de los casos. Por tanto, para que una operación unión $r \cup s$ sea válida hay que exigir que se cumplan dos condiciones:

nombre-cliente
Abril
Fernández
Gómez
González
López
Pérez
Rupérez
Santos
Sotoca
Valdivieso

FIGURA 3.12. Nombres de todos los clientes que tienen un préstamo o una cuenta.

1. Las relaciones r y s deben ser de la misma aridad. Es decir, deben tener el mismo número de atributos.
2. Los dominios de los atributos i-ésimos de r y de s deben ser iguales para todo i.

Téngase en cuenta que r y s pueden ser, en general, relaciones temporales que sean resultado de expresiones del álgebra relacional.

3.2.1.5. La operación diferencia de conjuntos

La operación **diferencia de conjuntos**, denotada por –, permite buscar las tuplas que estén en una relación pero no en la otra. La expresión $r – s$ da como resultado una relación que contiene las tuplas que están en r pero no en s.

Se pueden buscar todos los clientes del banco que tienen abierta una cuenta pero no tienen concedido ningún préstamo escribiendo

$$\Pi_{nombre\text{-}cliente}\,(impositor) - \Pi_{nombre\text{-}cliente}\,(prestatario)$$

La relación resultante de esta consulta aparece en la Figura 3.13.

Como en el caso de la operación unión, hay que asegurarse de que las diferencias de conjuntos se realicen entre relaciones *compatibles*. Por tanto, para que una

nombre-cliente
Abril
González
Rupérez

FIGURA 3.13. Clientes con cuenta abierta pero sin préstamo concedido.

operación diferencia de conjuntos $r - s$ sea válida hay que exigir que las relaciones r y s sean de la misma aridad y que los dominios de los atributos i-ésimos de r y s sean iguales.

3.2.1.6. La operación producto cartesiano

La operación **producto cartesiano**, denotada por un aspa (×), permite combinar información de cualesquiera dos relaciones. El producto cartesiano de las relaciones r_1 y r_2 como $r_1 \times r_2$.

Recuérdese que las relaciones se definen como subconjuntos del producto cartesiano de un conjunto de dominios. A partir de esta definición ya se debe tener una intuición sobre la definición de la operación producto cartesiano. Sin embargo, dado que el mismo nombre de atributo puede aparecer tanto en r_1 como en r_2, hay que crear un esquema de denominaciones para distinguir entre ambos atributos. En este caso se logra adjuntando al atributo el nombre de la relación de la que proviene originalmente. Por ejemplo, el esquema de relación de $r = prestatario \times préstamo$ es

(*prestatario.nombre-cliente, prestatario.número-préstamo, préstamo.nombre-sucursal, préstamo.número-préstamo, préstamo.importe*)

Con este esquema se puede distinguir entre *prestatario.número-préstamo* y *préstamo.número-préstamo*. Para los atributos que sólo aparecen en uno de los dos esquemas se suele omitir el prefijo con el nombre de la relación. Esta simplificación no genera ambigüedad alguna. Por tanto, se puede escribir el esquema de relación de r como

(*nombre-cliente, prestatario.número-préstamo, nombre-sucursal, préstamo.número-préstamo, importe*)

El acuerdo de denominaciones precedente *exige* que las relaciones que sean argumentos de la operación producto cartesiano tengan nombres diferentes. Esta exigencia causa problemas en algunos casos, como cuando se desea calcular el producto cartesiano de una relación consigo misma. Se produce un problema similar si se utiliza el resultado de una expresión del álgebra relacional en un producto cartesiano, dado que hará falta un nombre para la relación para poder hacer referencia a sus atributos. En el Apartado 3.2.1.7 se verá la manera de evitar estos problemas utilizando una operación renombramiento.

Ahora que se conoce el esquema de relación de $r = prestatario \times préstamo$ hay que averiguar las tuplas que aparecerán en r. Como se podía imaginar, se crea una tupla de r a partir de cada par de tuplas posible: una de la relación *prestatario* y otra de la relación *préstamo*. Por tanto, r es una relación de gran tamaño, como se puede ver en la Figura 3.14, donde sólo se ha incluido una parte de las tuplas que forman parte de r.

Supóngase que se tienen n_1 tuplas en *prestatario* y n_2 tuplas en *préstamo*. Por tanto, hay $n_1 * n_2$ maneras de escoger un par de tuplas, una tupla de cada relación; por lo que hay $n_1 * n_2$ tuplas en r. En concreto, obsérvese que para algunas tuplas t de r puede ocurrir que t[*prestatario.número-préstamo*] ≠ t[*préstamo.número-préstamo*].

En general, si se tienen las relaciones r_1 (R_1) y r_2 (R_2), $r_1 \times r_2$ es una relación cuyo esquema es la concatenación de R_1 y de R_2. La relación R contiene todas las tuplas t para las que hay unas tuplas t_1 en r_1 y t_2 en r_2 para las que $t[R_1] = t_1[R_1]$ y $t[R_2] = t_2[R_2]$.

Supóngase que se desea averiguar los nombres de todos los clientes que tienen concedido un préstamo en la sucursal de Navacerrada. Se necesita para ello información de las relaciones *préstamo* y *prestatario*. Si se escribe

$$\sigma_{\text{nombre-sucursal} = \text{«Navacerrada»}} (prestatario \times préstamo)$$

entonces el resultado es la relación mostrada en la Figura 3.15. Se tiene una relación que sólo atañe a la sucursal de Navacerrada. Sin embargo, la columna *nombre-cliente* puede contener clientes que no tengan concedido ningún préstamo en la sucursal de Navacerrada. (Si no se ve el motivo por el que esto es cierto, recuérdese que el producto cartesiano toma todos los emparejamientos posibles de una tupla de *prestatario* con una tupla de *préstamo*.)

Dado que la operación producto cartesiano asocia *todas* las tuplas de *préstamo* con todas las tuplas de *prestatario*, se sabe que, si un cliente tiene concedido un préstamo en la sucursal de Navacerrada, hay alguna tupla de *prestatario* × *préstamo* que contiene su nombre y que *prestatario.número-préstamo* = *préstamo.número-préstamo*. Por tanto, si escribimos

$$\sigma_{prestatario.número\text{-}préstamo = préstamo.número\text{-}préstamo} (\sigma_{nombre\text{-}sucursal = \text{«Navacerrada»}} (prestatario \times préstamo))$$

sólo se obtienen las tuplas de *prestatario* × *préstamo* que corresponden a los clientes que tienen concedido un préstamo en la sucursal de Navacerrada.

Finalmente, dado que sólo se desea obtener *nombre-cliente*, se realiza una proyección:

$$\Pi_{nombre\text{-}cliente} (\sigma_{prestatario.número\text{-}préstamo = préstamo.número\text{-}préstamo} (\sigma_{nombre\text{-}sucursal = \text{«Navacerrada»}} (prestatario \times préstamo)))$$

El resultado de esta expresión se muestra en la Figura 3.16 y es la respuesta correcta a la consulta formulada.

3.2.1.7. La operación renombramiento

A diferencia de las relaciones de la base de datos, los resultados de las expresiones de álgebra relacional no tienen un nombre que se pueda utilizar para referirse a ellas. Resulta útil poder ponerles nombre; el operador

nombre-cliente	*prestatario.número-préstamo*	*préstamo.número-préstamo*	*nombre-sucursal*	*importe*
Santos	P-17	P-11	Collado Mediano	900
Santos	P-17	P-14	Centro	1.500
Santos	P-17	P-15	Navacerrada	1.500
Santos	P-17	P-16	Navacerrada	1.300
Santos	P-17	P-17	Centro	1.000
Santos	P-17	P-23	Moralzarzal	2.000
Santos	P-17	P-93	Becerril	500
Gómez	P-23	P-11	Collado Mediano	900
Gómez	P-23	P-14	Centro	1.500
Gómez	P-23	P-15	Navacerrada	1.500
Gómez	P-23	P-16	Navacerrada	1.300
Gómez	P-23	P-17	Centro	1.000
Gómez	P-23	P-23	Moralzarzal	2.000
Gómez	P-23	P-93	Becerril	500
López	P-15	P-11	Collado Mediano	900
López	P-15	P-14	Centro	1.500
López	P-15	P-15	Navacerrada	1.500
López	P-15	P-16	Navacerrada	1.300
López	P-15	P-17	Centro	1.000
López	P-15	P-23	Moralzarzal	2.000
López	P-15	P-93	Becerril	500
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
Valdivieso	P-17	P-11	Collado Mediano	900
Valdivieso	P-17	P-14	Centro	1.500
Valdivieso	P-17	P-15	Navacerrada	1.500
Valdivieso	P-17	P-16	Navacerrada	1.300
Valdivieso	P-17	P-17	Centro	1.000
Valdivieso	P-17	P-23	Moralzarzal	2.000
Valdivieso	P-17	P-93	Becerril	500
Fernández	P-16	P-11	Collado Mediano	900
Fernández	P-16	P-14	Centro	1.500
Fernández	P-16	P-15	Navacerrada	1.500
Fernández	P-16	P-16	Navacerrada	1.300
Fernández	P-16	P-17	Centro	1.000
Fernández	P-16	P-23	Moralzarzal	2.000
Fernández	P-16	P-93	Becerril	500

FIGURA 3.14. Resultado de *prestatario* × *préstamo*.

nombre-cliente	*prestatario.número-préstamo*	*préstamo.número-préstamo*	*nombre-sucursal*	*importe*
Santos	P-17	P-15	Navacerrada	1.500
Santos	P-17	P-16	Navacerrada	1.300
Gómez	P-23	P-15	Navacerrada	1.500
Gómez	P-23	P-16	Navacerrada	1.300
López	P-15	P-15	Navacerrada	1.500
López	P-15	P-16	Navacerrada	1.300
Sotoca	P-14	P-15	Navacerrada	1.500
Sotoca	P-14	P-16	Navacerrada	1.300
Pérez	P-93	P-15	Navacerrada	1.500
Pérez	P-93	P-16	Navacerrada	1.300
Gómez	P-11	P-15	Navacerrada	1.500
Gómez	P-11	P-16	Navacerrada	1.300
Valdivieso	P-17	P-15	Navacerrada	1.500
Valdivieso	P-17	P-16	Navacerrada	1.300
Fernández	P-16	P-15	Navacerrada	1.500
Fernández	P-16	P-16	Navacerrada	1.300

FIGURA 3.15. Resultado de $\sigma_{\text{nombre-sucursal = «Navacerrada»}}$ (*prestatario* × *préstamo*).

nombre-cliente
Férnandez
López

FIGURA 3.16. Resultado de $\Pi_{nombre\text{-}cliente}\,(\sigma_{prestatario.número\text{-}préstamo\,=\,préstamo.número\text{-}préstamo}\,(\sigma_{nombre\text{-}sucursal\,=\,\text{«Navacerrada»}}\,(prestatario \times préstamo)))$.

renombramiento, denotado por la letra griega rho minúscula (ρ), permite realizar esta tarea. Dada una expresión E del álgebra relacional, la expresión

$$\rho_x\,(E)$$

devuelve el resultado de la expresión E con el nombre x.

Las relaciones r por sí mismas se consideran expresiones (triviales) del álgebra relacional. Por tanto, también se puede aplicar la operación renombramiento a una relación r para obtener la misma relación con un nombre nuevo.

Otra forma de la operación renombramiento es la siguiente. Supóngase que una expresión del álgebra relacional E tiene aridad n. Por tanto, la expresión

$$\rho_{x\,(A_1,\,A_2,\,\ldots\,A_n)}\,(E)$$

devuelve el resultado de la expresión E con el nombre x y con los atributos con el nombre cambiado a $A_1, A_2, \ldots, A_n$.

Para ilustrar el uso del renombramiento de las relaciones, considérese la consulta «Buscar el máximo saldo de cuenta del banco». La estrategia empleada para obtener el resultado es 1) calcular una relación intermedia consistente en los saldos que *no* son el máximo y 2) realizar la diferencia entre la relación $\Pi_{saldo}\,(cuenta)$ y la relación intermedia recién calculada.

Paso 1: Para calcular la relación intermedia hay que comparar los valores de los saldos de todas las cuentas. Esta comparación se puede hacer calculando el producto cartesiano *cuenta* × *cuenta* y formando una selección para comparar el valor de cualesquiera dos saldos que aparezcan en una tupla. En primer lugar hay que crear un mecanismo para distinguir entre los dos atributos *saldo*. Se utilizará la operación renombramiento para cambiar el nombre de una referencia a la relación cuenta; así, se puede hacer referencia dos veces a la relación sin ambigüedad alguna.

La relación temporal que se compone de los saldos que no son el máximo puede escribirse ahora como

$$\Pi_{cuenta.saldo}\,(\sigma_{cuenta.saldo\,<\,d.saldo}\,(cuenta \times \rho_d\,(cuenta)))$$

Esta expresión proporciona los saldos de la relación *cuenta* para los que aparece un saldo mayor en alguna parte de la relación *cuenta* (cuyo nombre se ha cambiado a *d*). El resultado contiene todos los saldos *salvo* el máximo. Esta relación se muestra en la Figura 3.17.

saldo
500
400
700
750
350

FIGURA 3.17. Resultado de la subexpresión $\Pi_{cuenta\text{-}saldo}\,(\sigma_{cuenta\text{-}saldo\,<\,d.saldo}\,(cuenta \times \rho_d\,(cuenta)))$.

Paso 2: La consulta para averiguar el máximo saldo de cuenta del banco puede escribirse de la manera siguiente:

$$\Pi_{saldo}\,(cuenta) - \Pi_{cuenta.saldo}\,(\sigma_{cuenta.saldo\,<\,d.saldo}\,(cuenta \times \rho_d\,(cuenta)))$$

En la Figura 3.18 se muestra el resultado de esta consulta.

Considérese la siguiente consulta como un nuevo ejemplo de la operación renombramiento: «Averiguar los nombres de todos los clientes que viven en la misma calle y en la misma ciudad que Gómez». Se puede obtener la calle y la ciudad en la que vive Gómez escribiendo

$$\Pi_{calle\text{-}cliente,\,ciudad\text{-}cliente}\,(\sigma_{nombre\text{-}cliente\,=\,\text{«Gómez»}}\,(cliente))$$

Sin embargo, para hallar a otros clientes que vivan en esa calle y en esa ciudad hay que hacer referencia por segunda vez a la relación *cliente*. En la consulta siguiente se utiliza la operación renombramiento sobre la expresión anterior para darle al resultado el nombre *dirección-Gómez* y para cambiar el nombre de los atributos a *calle* y *ciudad* en lugar de *calle-cliente* y *ciudad-cliente*:

$$\Pi_{cliente.nombre\text{-}cliente}\,(\sigma_{cliente.calle\text{-}cliente\,=\,dirección\text{-}Gómez\,\wedge\,cliente.ciudad\text{-}cliente\,=\,dirección\text{-}Gómez.\,ciudad}\,(cliente \times \rho_{dirección\text{-}Gómez\,(calle,\,ciudad)}\,(\Pi_{calle\text{-}cliente,\,ciudad\text{-}cliente}\,(\sigma_{nombre\text{-}cliente\,=\,\text{«Gómez»}}\,(cliente)))))$$

El resultado de esta consulta, cuando se aplica a la relación *cliente* de la Figura 3.4, se muestra en la Figura 3.19.

La operación renombramiento no es estrictamente necesaria, dado que es posible utilizar una notación posicional para los atributos. Se pueden nombrar los atributos de una relación de manera implícita utilizando una notación posicional, donde \$1, \$2, … hagan refe-

saldo
900

FIGURA 3.18. Saldo máximo de las cuentas del banco.

nombre-cliente
Gómez
Pérez

FIGURA 3.19. Los clientes que viven en la misma calle y en la misma ciudad que Gómez.

rencia al primer atributo, al segundo, etcétera. La notación posicional también se aplica a los resultados de las operaciones del álgebra relacional. La siguiente expresión del álgebra relacional ilustra el uso de la notación posicional con el operador unario σ:

$$\sigma_{\$2=\$3}(R \times R)$$

Si una operación binaria necesita distinguir entre las dos relaciones que son sus operandos, se puede utilizar una notación posicional parecida para los nombres de las relaciones. Por ejemplo, $\$R1$ puede hacer referencia al primer operando y $\$R2$, al segundo. Sin embargo, la notación posicional no resulta conveniente para las personas, dado que la posición del atributo es un número en vez de un nombre de atributo fácil de recordar. Por tanto, en este libro no se utiliza la notación posicional.

3.2.2. Definición formal del álgebra relacional

Las operaciones que se vieron en el Apartado 3.2.1 permiten dar una definición completa de las expresiones del álgebra relacional. Las expresiones fundamentales del álgebra relacional se componen de alguna de las siguientes:

- Una relación de la base de datos
- Una relación constante

Una relación constante se escribe listando sus tuplas entre llaves ({}), por ejemplo {(C-101,Centro,500) (C-215, Becerril, 700)}.

Las expresiones generales del álgebra relacional se construyen a partir de subexpresiones menores. Sean E_1 y E_2 expresiones de álgebra relacional. Todas las siguientes son expresiones del álgebra relacional:

- $E_1 \cup E_2$
- $E_1 - E_2$
- $E_1 \times E_2$
- $\sigma_P(E_1)$, donde P es un predicado de atributos de E_1
- $\Pi_S(E_1)$, donde S es una lista que se compone de algunos de los atributos de E_1
- $\rho_x(E_1)$, donde x es el nuevo nombre del resultado de E_1.

3.2.3. Otras operaciones

Las operaciones fundamentales del álgebra relacional son suficientes para expresar cualquier consulta del álgebra relacional[1]. Sin embargo, si uno se limita únicamente a las operaciones fundamentales, algunas consultas habituales resultan de expresión intrincada. Por tanto, se definen otras operaciones que no añaden potencia al álgebra, pero que simplifican las consultas habituales. Para cada operación nueva se facilita una expresión equivalente utilizando sólo las operaciones fundamentales.

3.2.3.1. La operación intersección de conjuntos

La primera operación adicional del álgebra relacional que se definirá es la **intersección de conjuntos** ($\cap$). Supóngase que se desea averiguar todos los clientes que tienen un préstamo concedido y una cuenta abierta. Utilizando la intersección de conjuntos se puede escribir

$$\Pi_{nombre\text{-}cliente}(prestatario) \cap \Pi_{nombre\text{-}cliente}(impositor)$$

La relación resultante de esta consulta aparece en la Figura 3.20.

Obsérvese que se puede volver a escribir cualquier expresión del álgebra relacional utilizando la intersección de conjuntos sustituyendo la operación intersección por un par de operaciones de diferencia de conjuntos, de la manera siguiente:

$$r \cap s = r - (r - s)$$

Por tanto, la intersección de conjuntos no es una operación fundamental y no añade potencia al álgebra relacional. Sencillamente, es más conveniente escribir $r \cap s$ que $r - (r - s)$.

3.2.3.2. La operación reunión natural

Suele resultar deseable simplificar ciertas consultas que exigen un producto cartesiano. Generalmente, las consultas que implican un producto cartesiano incluyen un operador selección sobre el resultado del producto cartesiano. Considérese la consulta «Hallar los nombres de todos los clientes que tienen concedido un préstamo en el banco y averiguar el importe del mismo». En primer lugar se calcula el producto cartesiano de las relaciones *prestatario* y *préstamo*. Luego, se seleccionan las tuplas que sólo atañen al mismo *número-préstamo*, seguidas por la proyección de *nombre-cliente*, *número-préstamo* e *importe* resultantes:

$$\Pi_{nombre\text{-}cliente,\ préstamo.número\text{-}préstamo,\ importe}\left(\sigma_{prestatario.número\text{-}préstamo\ =\ préstamo.número\text{-}préstamo}(prestatario \times préstamo)\right)$$

[1] En el Apartado 3.3 se introducen las operaciones que extienden la potencia del álgebra relacional al tratamiento de los valores nulos y los valores de agregación.

nombre-cliente
Gómez
Pérez
Santos

FIGURA 3.20. Clientes con una cuenta abierta y un préstamo en el banco

La *reunión natural* es una operación binaria que permite combinar ciertas selecciones y un producto cartesiano en una sola operación. Se denota por el símbolo de la «reunión» $\bowtie$. La operación reunión natural forma un producto cartesiano de sus dos argumentos, realiza una selección forzando la igualdad de los atributos que aparecen en ambos esquemas de relación y, finalmente, elimina los atributos duplicados.

Aunque la definición de la reunión natural es compleja, la operación es sencilla de aplicar. Como ilustración, considérese nuevamente el ejemplo «Averiguar los nombres de todos los clientes que tienen concedido un préstamo en el banco y averiguar su importe». Esta consulta puede expresarse utilizando la reunión natural de la manera siguiente:

$$\Pi_{\textit{nombre-cliente, número-préstamo, importe}} (\textit{prestatario} \bowtie \textit{préstamo})$$

Dado que los esquemas de *prestatario* y de *préstamo* (es decir, *Esquema-prestatario* y *Esquema-préstamo*) tienen en común el atributo *número-préstamo*, la operación reunión natural sólo considera los pares de tuplas que tienen el mismo valor de *número-préstamo*. Esta operación combina cada uno de estos pares en una sola tupla en la unión de los dos esquemas (es decir, *nombre-cliente*, *nombre-sucursal*, *número-préstamo*, *importe*). Después de realizar la proyección, se obtiene la relación mostrada en la Figura 3.21.

Considérense dos esquemas de relación R y S que son, por supuesto, listas de nombres de atributos. Si se consideran los esquemas como *conjuntos*, en vez de como listas, se pueden denotar los nombres de los atributos que aparecen tanto en R como en S mediante $R \cap S$, y los nombres de los atributos que aparecen en R, en S o en ambos mediante $R \cup S$. De manera parecida, los nombres de los atributos que aparecen en R pero no en S se denotan por $R - S$, mientras que $S - R$ denota los nombres de los atributos que aparecen en S pero no en R. Obsérvese que las operaciones unión, intersección y diferencia aquí operan sobre conjuntos de atributos, y no sobre relaciones.

Ahora se está preparado para una definición formal de la reunión natural. Considérense dos relaciones $r(R)$ y $s(S)$. La **reunión natural** de r y de s, denotada por $r \bowtie s$ es una relación del esquema $R \cup S$ definida formalmente de la manera siguiente:

$$r \bowtie \text{s} = \Pi_{R \cup S} (\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n} r \times s)$$

donde $R \cap S = \{A_1, A_2, \ldots, A_n\}$.

Como la reunión natural es fundamental para gran parte de la teoría y de la práctica de las bases de datos relacionales, se ofrecen varios ejemplos de su uso.

- Hallar los nombres de todas las sucursales con clientes que tienen una cuenta abierta en el banco y que viven en Peguerinos.

  $$\Pi_{\textit{nombre-sucursal}} (\sigma_{\textit{ciudad-cliente} = \text{«Peguerinos»}} (\textit{cliente} \bowtie \textit{cuenta} \bowtie \textit{impositor}))$$

  La relación resultante de esta consulta aparece en la Figura 3.22.

  Obsérvese que se escribió *cliente* $\bowtie$ *cuenta* $\bowtie$ *impositor* sin añadir paréntesis para especificar el orden en que se deben ejecutar las operaciones reunión natural de las tres relaciones. En el caso anterior hay dos posibilidades:

  — (*cliente* $\bowtie$ *cuenta*) $\bowtie$ *impositor*
  — *cliente* $\bowtie$ (*cuenta* $\bowtie$ *impositor*)

  No se especificó la expresión deseada porque las dos son equivalentes. Es decir, la reunión natural es **asociativa**.

- Hallar todos los clientes que tienen una cuenta abierta *y* un préstamo concedido en el banco.

  $$\Pi_{\textit{nombre-cliente}} (\textit{prestatario} \bowtie \textit{impositor})$$

  Obsérvese que en el Apartado 3.2.3.1 se escribió una expresión para esta consulta utilizando la intersección de conjuntos. Aquí se repite esa expresión.

$$\Pi_{\textit{nombre-cliente}} (\textit{prestatario}) \cap \Pi_{\textit{nombre-cliente}} (\textit{impositor})$$

nombre-cliente	*número-préstamo*	*importe*
Fernández	P-16	1.300
Gómez	P-23	2.000
Gómez	P-11	900
López	P-15	1.500
Pérez	P-93	500
Santos	P-17	1.000
Sotoca	P-14	1.500
Valdivieso	P-17	1.000

FIGURA 3.21. Resultado de $\Pi_{\textit{nombre-cliente, número-préstamo, importe}} (\textit{prestatario} \bowtie \textit{préstamo})$.

nombre-sucursal
Galapagar
Navacerrada

FIGURA 3.22. Resultado de $\Pi_{\textit{nombre-sucursal}} (\sigma_{\textit{ciudad-cliente} = \text{«Peguerinos»}} (\textit{cliente} \bowtie \textit{cuenta} \bowtie \textit{impositor}))$.

La relación resultante de esta consulta se mostró anteriormente en la Figura 3.20. Este ejemplo ilustra una realidad común del álgebra relacional: se pueden escribir varias expresiones del álgebra relacional equivalentes que sean bastante diferentes entre sí.

- Sean $r(R)$ y $s(S)$ relaciones sin atributos en común; es decir, $R \cap S = \emptyset$. ($\emptyset$ denota el conjunto vacío.) Por tanto, $r \bowtie s = r \times s$.

La operación *reunión zeta* es una extensión de la operación reunión natural que permite combinar una selección y un producto cartesiano en una sola operación. Considérense las relaciones $r(R)$ y $s(S)$, y sea θ un predicado de los atributos del esquema $R \cup S$. La operación **reunión zeta** $r \bowtie_\theta s$ se define de la manera siguiente:

$$r \bowtie_\theta s = \sigma_\theta (r \times s)$$

3.2.3.3. La operación división

La operación **división**, denotada por ÷, resulta adecuada para las consultas que incluyen la expresión «para todos». Supóngase que se desea hallar a todos los clientes que tengan abierta una cuenta en *todas* las sucursales ubicadas en Arganzuela. Se pueden obtener todas las sucursales de Arganzuela mediante la expresión

$$r_1 = \Pi_{nombre\text{-}sucursal} (\sigma_{ciudad\text{-}sucursal = \text{«Arganzuela»}} (sucursal))$$

La relación resultante de esta expresión aparece en la Figura 3.23.

Se pueden encontrar todos los pares (*nombre-cliente*, *nombre-sucursal*) para los que el cliente tiene una cuenta en una sucursal escribiendo

$$r_2 = \Pi_{nombre\text{-}cliente,\ nombre\text{-}sucursal} (impositor \bowtie cuenta)$$

La Figura 3.24 muestra la relación resultante de esta expresión.

Ahora hay que hallar los clientes que aparecen en r_2 con los nombres de *todas* las sucursales de r_1. La operación que proporciona exactamente esos clientes es la operación división. La consulta se formula escribiendo

$$\Pi_{nombre\text{-}cliente,\ nombre\text{-}sucursal} (impositor \bowtie cuenta) \div \Pi_{nombre\text{-}sucursal} (\sigma_{ciudad\text{-}sucursal = \text{«Arganzuela»}} (sucursal))$$

El resultado de esta expresión es una relación que tiene el esquema (*nombre-cliente*) y que contiene la tupla (González).

nombre-sucursal
Centro
Galapagar

FIGURA 3.22. Resultado de $\Pi_{nombre\text{-}sucursal} (\sigma_{ciudad\text{-}sucursal = \text{«Arganzuela»}} (sucursal))$.

nombre-cliente	*nombre-sucursal*
Abril	Collado Mediano
Gómez	Becerril
González	Centro
González	Galapagar
López	Navacerrada
Rupérez	Moralzarzal
Santos	Galapagar
Valdivieso	Navacerrada

FIGURA 3.24. Resultado de $\Pi_{nombre\text{-}cliente,\ nombre\text{-}sucursal} (impositor \bowtie cuenta)$.

Formalmente, sean $r(R)$ y $s(S)$ relaciones y $S \subseteq R$; es decir, todos los atributos del esquema S están también en el esquema R. La relación $r \div s$ es una relación del esquema $R - S$ (es decir, del esquema que contiene todos los atributos del esquema R que no están en el esquema S). Una tupla t está en $r \div s$ si y sólo si se cumplen estas dos condiciones:

1. t está en $\Pi_{R-S} (r)$
2. Para cada tupla t_S de s hay una tupla t_r de r que cumple las dos condiciones siguientes:
   a. $t_r[S] = t_s[S]$
   b. $t_r[R - S] = t$

Puede resultar sorprendente descubrir que, dados una operación división y los esquemas de las relaciones, se puede, de hecho, definir la operación división en términos de las operaciones fundamentales. Sean $r(R)$ y $s(S)$ dadas, con $S \subseteq R$:

$$r \div s = \Pi_{R-S} (r) - \Pi_{R-S} ((\Pi_{R-S} (r) \times s) - \Pi_{R-S,S} (r))$$

Para comprobar que esta expresión es verdadera, obsérvese que $\Pi_{R-S} (r)$ da todas las tuplas t que cumplen la primera condición de la definición de la división. La expresión del lado derecho del operador diferencia de conjuntos,

$$\Pi_{R-S} ((\Pi_{R-S} (r) \times s) - \Pi_{R-S,S} (r)),$$

sirve para borrar esas tuplas que no cumplen la segunda condición de la definición de la división. Esto se logra de la manera siguiente. Considérese $\Pi_{R-S} (r) \times s$. Esta relación está en el esquema R y empareja cada tupla de $\Pi_{R-S} (r)$ con cada tupla de s. La expresión $\Pi_{R-S,S} (r)$ sólo reordena los atributos de r.

Por tanto, $(\Pi_{R-S} (r) \times s) - \Pi_{R-S,S} (r)$ genera los pares de tuplas de $\Pi_{R-S} (r)$ y de s que no aparecen en r. Si una tupla t_j está en

$$\Pi_{R-S} ((\Pi_{R-S} (r) \times s) - \Pi_{R-S,S} (r)),$$

hay alguna tupla t_s de s que no se combina con la tupla t_j para formar una tupla de r. Por tanto, t_j guarda un valor de los atributos $R - S$ que no aparece en $r \div s$. Estos valores son los que se eliminan de $\Pi_{R-S} (r)$.

3.2.3.4. La operación asignación

En ocasiones resulta conveniente escribir una expresión del álgebra relacional por partes utilizando la asignación a una variable de relación temporal. La operación **asignación**, denotada por ←, actúa de manera parecida a la asignación de los lenguajes de programación. Para ilustrar esta operación, considérese la definición de la división dada en el Apartado 3.2.3.3. Se puede escribir $r \div s$ como

$$temp1 \leftarrow \Pi_{R-S}(r)$$
$$temp2 \leftarrow \Pi_{R-S}((temp1 \times s) - \Pi_{R-S,S}(r))$$
$$resultado = temp1 - temp2$$

La evaluación de una asignación no hace que se muestre ninguna relación al usuario. Por el contrario, el resultado de la expresión a la derecha de ← se asigna a la variable relación a la izquierda de ←. Esta variable relación puede utilizarse en expresiones posteriores.

Con la operación asignación se pueden escribir las consultas como programas secuenciales consistentes en una serie de asignaciones seguida de una expresión cuyo valor se muestra como resultado de la consulta. En las consultas del álgebra relacional la asignación siempre debe hacerse a una variable de relación intermedia. Las asignaciones a relaciones permanentes constituyen una modificación de la base de datos. Este asunto se discutirá en el Apartado 3.4. Obsérvese que la operación asignación no añade potencia alguna al álgebra. Resulta, sin embargo, una manera conveniente de expresar las consultas complejas.

3.3. OPERACIONES DEL ÁLGEBRA RELACIONAL EXTENDIDA

Las operaciones básicas del álgebra relacional se han ampliado de varias maneras. Una ampliación sencilla es permitir operaciones aritméticas como parte de la proyección. Una ampliación importante es permitir *operaciones de agregación*, como el cálculo de la suma de los elementos de un conjunto, o su media. Otra ampliación importante es la operación *reunión externa*, que permite a las expresiones del álgebra relacional trabajar con los valores nulos que modelan la información que falta.

3.3.1. Proyección generalizada

La operación **proyección generalizada** amplía la operación proyección permitiendo que se utilicen funciones aritméticas en la lista de proyección. La operación proyección generalizada tiene la forma

$$\Pi_{F_1, F_2, \ldots, F_n}(E)$$

donde E es cualquier expresión del álgebra relacional y $F_1, F_2, \ldots, F_n$ son expresiones aritméticas que incluyen constantes y atributos en el esquema de E. Como caso especial la expresión aritmética puede ser simplemente un atributo o una constante.

Por ejemplo, supóngase que se dispone de una relación *información-crédito*, como se muestra en la Figura 3.25, que da el límite de crédito y el importe dispuesto hasta el momento presente (el *saldo-crédito* de la cuenta). Si se desea averiguar el importe disponible por cada persona, se puede escribir la expresión siguiente:

$$\Pi_{\textit{nombre-cliente, límite - saldo-crédito}}(\textit{información-crédito})$$

El atributo resultante de la expresión *límite – saldo-crédito* no tiene un nombre. Se puede aplicar la operación renombramiento al resultado de la proyección generalizada para darle un nombre. Como conveniencia notacional, el renombramiento de atributos se puede combinar con la proyección generalizada como se ilustra a continuación:

$$\Pi_{\textit{nombre-cliente, (límite – saldo-crédito)}\ \textbf{as}\ \textit{crédito-disponible}}(\textit{información-crédito})$$

Al segundo atributo de esta proyección generalizada se le ha dado el nombre *crédito-disponible*. En la Figura 3.26 se muestra el resultado de aplicar esta expresión a la relación de la Figura 3.25.

3.3.2. Funciones de agregación

Las **funciones de agregación** son funciones que toman una colección de valores y devuelven como resultado un único valor. Por ejemplo, la función de agregación

nombre-cliente	*límite*	*saldo-crédito*
Gómez	2.000	400
López	1.500	1.500
Pérez	2.000	1.750
Santos	6.000	700

FIGURA 3.25. La relación *información-crédito*.

nombre-cliente	*crédito-disponible*
Gómez	1.600
López	0
Pérez	250
Santos	5.300

FIGURA 3.26. Resultado de $\Pi_{\textit{nombre-cliente, (límite – saldo-crédito)}\ \textbf{as}\ \textit{crédito-disponible}}(\textit{información-crédito})$.

sum toma un conjunto de valores y devuelve la suma de los mismos. Por tanto, la función **sum** aplicada a la colección

$$\{1, 1, 3, 4, 4, 11\}$$

devuelve el valor 24. La función de agregación **avg** devuelve la media de los valores. Cuando se aplica al conjunto anterior devuelve el valor 4. La función de agregación **count** devuelve el número de elementos del conjunto, y devolvería 6 en el caso anterior. Otras funciones de agregación habituales son **min** y **max**, que devuelven el valor mínimo y el máximo de la colección; en el ejemplo anterior devuelven 1 y 11, respectivamente.

Las colecciones en las que operan las funciones de agregación pueden tener valores repetidos; el orden en el que aparezcan los valores no tiene importancia. Estas colecciones se denominan **multiconjuntos**. Los conjuntos son un caso especial de los multiconjuntos, en los que sólo hay una copia de cada elemento.

Para ilustrar el concepto de agregación se utilizará la relación *trabajo-por-horas* descrita en la Figura 3.27, que muestra los empleados a tiempo parcial. Supóngase que se desea averiguar la suma total de los sueldos de los empleados del banco a tiempo parcial. La expresión del álgebra relacional para esta consulta es:

$$\mathcal{G}_{\textbf{sum}(sueldo)}\,(trabajo\text{-}por\text{-}horas)$$

El símbolo $\mathcal{G}$ es la letra G en el tipo de letra caligráfico; se lee «G caligráfica». La operación del álgebra relacional $\mathcal{G}$ significa que se debe aplicar agregación, y el subíndice indica la operación de agregación a aplicar. El resultado de la expresión anterior es una relación con un único atributo, que contiene una sola fila con un valor correspondiente a la suma de los sueldos de todos los trabajadores que trabajan en el banco a tiempo parcial.

Hay casos en los que se deben borrar los valores repetidos antes de calcular una función de agregación. Si se desean borrar los valores repetidos hay que utilizar los mismos nombres de funciones que antes, con la cadena de texto «**distinct**» precedida de un guión añadida al final del nombre de la función (por ejemplo, **count-distinct**). Un ejemplo se da en la consulta «Averiguar el número de sucursales que aparecen en la relación *trabajo-por-horas*». En este caso, el nombre de cada sucursal sólo se cuenta una vez, independientemente del número de empleados que trabajen en la misma. Esta consulta se escribe de la manera siguiente:

$$\mathcal{G}_{\textbf{count-distinct}(\text{nombre-}sucursal)}\,(trabajo\text{-}por\text{-}horas)$$

Para la relación mostrada en la Figura 3.27 el resultado de esta consulta es el valor 3.

Supóngase que se desea hallar la suma total de sueldos de todos los empleados a tiempo parcial en cada sucursal del banco por separado, en lugar de hallar la suma de sueldos de todo el banco. Para ello hay que dividir la relación *trabajo-por-horas* en **grupos** basados en la sucursal y aplicar la función de agregación a cada grupo.

La expresión siguiente obtiene el resultado deseado utilizando el operador de agregación $\mathcal{G}$:

$$_{nombre\text{-}sucursal}\mathcal{G}_{\textbf{sum}(sueldo)}\,(trabajo\text{-}por\text{-}horas)$$

El atributo *nombre-sucursal* subíndice a la izquierda de G indica que la relación de entrada *trabajo-por-horas* debe dividirse en grupos de acuerdo con el valor de *nombre-sucursal*. Los grupos resultantes se muestran en la Figura 3.28. La expresión **sum**(*sueldo*) en el subíndice derecho de $\mathcal{G}$ indica que, para cada grupo de tuplas (es decir, para cada sucursal) hay que aplicar la función de agregación **sum** al conjunto de valores del atributo *sueldo*. La relación resultante consiste en las tuplas con el nombre de la sucursal y la suma de los sueldos de la sucursal, como se muestra en la Figura 3.29.

La forma general de la **operación de agregación** $\mathcal{G}$ es la siguiente:

$$G_1, G_2, \ldots, G_n\,\mathcal{G}_{F_1(A_1),\,F_2(A_2),\,\ldots,\,F_m(A_m)}\,(E)$$

donde E es cualquier expresión del álgebra relacional; $G_1, G_2, \ldots, G_n$ constituye una lista de atributos que indican cómo se realiza la agrupación, cada F_i es una función de agregación y cada A_i es el nombre de un atributo. El significado de la operación se define de la manera siguiente. Las tuplas en el resultado de la expresión E se dividen en grupos tales que

nombre-empleado	*nombre-sucursal*	*sueldo*
González	Centro	1.500
Díaz	Centro	1.300
Jiménez	Centro	2.500
Catalán	Leganés	1.600
Cana	Leganés	1.500
Cascallar	Navacerrada	5.300
Fernández	Navacerrada	1.500
Ribera	Navacerrada	1.300

FIGURA 3.27. La relación *trabajo-por-horas*.

nombre-empleado	*nombre-sucursal*	*sueldo*
González	Centro	1.500
Díaz	Centro	1.300
Jiménez	Centro	2.500
Catalán	Leganés	1.600
Cana	Leganés	1.500
Cascallar	Navacerrada	5.300
Fernández	Navacerrada	1.500
Ribera	Navacerrada	1.300

FIGURA 3.28. La relación *trabajo-por-horas* después de la agrupación.

nombre-sucursal	*suma de sueldos*
Centro	5.300
Leganés	3.100
Navacerrada	8.100

FIGURA 3.29. Resultado de $_{nombre\text{-}sucursal}\mathcal{G}_{\textbf{sum}(\text{sueldo})}$ (*trabajo-por-horas*).

1. Todas las tuplas del grupo tienen los mismos valores para $G_1, G_2, \ldots, G_n$.
2. Las tuplas de grupos diferentes tienen valores diferentes para $G_1, G_2, \ldots, G_n$.

Por tanto, los grupos pueden identificarse por el valor de los atributos $G_1, G_2, \ldots, G_n$. Para cada grupo $(g_1, g_2, \ldots, g_n)$ el resultado tiene una tupla $(g_1, g_2, \ldots, g_n, a_1, a_2, \ldots, a_m)$ donde, para cada i, a_i es el resultado de aplicar la función de agregación F_i al multiconjunto de valores del atributo A_i en el grupo.

Como caso especial de la operación de agregación, la lista de atributos $G_1, G_2, \ldots, G_n$ puede estar vacía, en cuyo caso sólo hay un grupo que contiene todas las tuplas de la relación. Esto corresponde a la agregación sin agrupación.

Volviendo al ejemplo anterior, si se deseara averiguar el sueldo máximo de los empleados a tiempo parcial de cada oficina, además de la suma de los sueldos, habría que escribir la expresión

$$_{nombre\text{-}sucursal}\mathcal{G}_{\textbf{sum}(sueldo),\ \textbf{max}(sueldo)}\ (trabajo\text{-}por\text{-}horas)$$

Como en la proyección generalizada, el resultado de una operación de agregación no tiene nombre. Se puede aplicar la operación renombramiento al resultado para darle un nombre. Como conveniencia notacional, los atributos de una operación de agregación se pueden renombrar como se indica a continuación:

$$_{nombre\text{-}sucursal}\mathcal{G}_{\textbf{sum}(sueldo)\ \textbf{as}\ suma\text{-}sueldo,\textbf{max}(sueldo)\ \textbf{as}\ sueldo\text{-}máximo}\ (trabajo\text{-}por\text{-}horas)$$

El resultado de la expresión se muestra en la Figura 3.30.

3.3.3. Reunión externa

La operación **reunión externa** es una ampliación de la operación reunión para trabajar con la información que falta. Supóngase que se dispone de relaciones con los siguientes esquemas, que contienen datos de empleados a tiempo completo:

empleado (*nombre-empleado, calle, ciudad*)
trabajo-a-tiempo-completo (*nombre-empleado, nombre-sucursal, sueldo*)

Considérense las relaciones *empleado* y *trabajo-a-tiempo-completo* mostradas en la Figura 3.31. Supóngase que se desea generar una única relación con toda la información (calle, ciudad, nombre de la sucursal y sueldo) de los empleados a tiempo completo. Un posible enfoque sería utilizar la operación reunión natural de la manera siguiente:

$$empleado \bowtie trabajo\text{-}a\text{-}tiempo\text{-}completo$$

El resultado de esta expresión se muestra en la Figura 3.32. Obsérvese que se ha perdido la información sobre la calle y la ciudad de residencia de Gómez, dado que la tupla que describe a Gómez no está presente en la relación *trabajo-a-tiempo-completo*; de manera parecida, se ha perdido la información sobre el nombre de la sucursal y sobre el sueldo de Barea, dado que la tupla que describe a Barea no está presente en la relación *empleado*.

Se puede utilizar la operación reunión externa para evitar esta pérdida de información. En realidad, esta operación tiene tres formas diferentes: *reunión externa por la izquierda*, denotada por $\mathbin{⟕}$; *reunión externa por la derecha*, denotada por $\mathbin{⟖}$ y *reunión externa completa*, denotada por $\mathbin{⟗}$. Las tres formas de la reunión externa calculan la reunión y añaden tuplas adicionales al resultado de la misma. El resultado de las expresiones *empleado* $\bowtie$ *trabajo-a-tiempo-completo*, *empleado* $\mathbin{⟖}$ *trabajo-a-tiempo-completo* y *empleado* $\mathbin{⟗}$ *trabajo-a-tiempo-completo* se muestra en las Figuras 3.33, 3.34 y 3.35, respectivamente.

La **reunión externa por la izquierda** ($\mathbin{⟕}$) toma todas las tuplas de la relación de la izquierda que no coincidan con ninguna tupla de la relación de la derecha, las rellena con valores nulos en todos los demás atributos de la relación de la derecha y las añade al resul-

nombre-sucursal	*suma-sueldo*	*sueldo-máximo*
Centro	5.300	2.500
Leganés	3.100	1.600
Navacerrada	8.100	5.300

FIGURA 3.30. Resultado de $_{nombre\text{-}sucursal}\mathcal{G}_{\textbf{sum}(\text{sueldo})\ \textbf{as}\ \text{suma-sueldo}\ \textbf{max}(\text{sueldo})\ \textbf{as}\ \text{sueldo-máximo}}$ (*trabajo-por-horas*).

nombre-empleado	*calle*	*ciudad*
Segura	Tebeo	La Loma
Domínguez	Viaducto	Villaconejos
Gómez	Bailén	Alcorcón
Valdivieso	Fuencarral	Móstoles

nombre-empleado	*nombre-sucursal*	*sueldo*
Segura	Majadahonda	1.500
Domínguez	Majadahonda	1.300
Barea	Fuenlabrada	5.300
Valdivieso	Fuenlabrada	1.500

FIGURA 3.31. Las relaciones *empleado* y *trabajo-a-tiempo-completo.*

nombre-empleado	*calle*	*ciudad*	*nombre-sucursal*	*sueldo*
Segura	Tebeo	La Loma	Majadahonda	1.500
Domínguez	Viaducto	Villaconejos	Majadahonda	1.300
Valdivieso	Fuencarral	Móstoles	Fuenlabrada	1.500

FIGURA 3.32. La relación *empleado* ⋈ *trabajo-a-tiempo-completo.*

tado de la reunión natural. En la Figura 3.33 la tupla (Gómez, Bailén, Alcorcón, *nulo*, *nulo*) es una tupla de este tipo. Toda la información de la relación de la izquierda se halla presente en el resultado de la reunión externa por la izquierda.

La **reunión externa por la derecha** (⟖) es simétrica de la reunión externa por la izquierda. Las tuplas de la relación de la derecha que no coincidan con ninguna tupla de la relación de la izquierda se rellenan con valores nulos y se añaden al resultado de la reunión natural. En la Figura 3.34 la tupla (Barea, *nulo*, *nulo*, Fuenlabrada, 5.300) es una tupla de este tipo. Por tanto, toda la información de la relación de la derecha se halla presente en el resultado de la reunión externa por la derecha.

nombre-empleado	*calle*	*ciudad*	*nombre-sucursal*	*sueldo*
Segura	Tebeo	La Loma	Majadahonda	1.500
Domínguez	Viaducto	Villaconejos	Majadahonda	1.300
Valdivieso	Fuencarral	Móstoles	Fuenlabrada	1.500
Gómez	Bailén	Alcorcón	*nulo*	*nulo*

FIGURA 3.33. Resultado de *empleado* ⟖ *trabajo-a-tiempo-completo.*

La **reunión externa completa** (⟗) realiza estas dos operaciones, rellenando las tuplas de la relación de la izquierda que no coincidan con ninguna tupla de la relación de la derecha y las tuplas de la relación de la derecha que no coincidan con ninguna tupla de la relación de la izquierda, y añadiéndolas al resultado de la reunión. En la Figura 3.35 se muestra el resultado de una reunión externa completa.

Puesto que las operaciones de reunión pueden generar resultados que contengan nulos, es necesario especificar cómo deben manejar estos valores las operaciones del álgebra relacional. El Apartado 3.3.4 aborda este aspecto.

Es interesante observar que las operaciones de reunión externa pueden expresar mediante las operaciones básicas del álgebra relacional. Por ejemplo, la opera-

nombre-empleado	*calle*	*ciudad*	*nombre-sucursal*	*sueldo*
Segura	Tebeo	La Loma	Majadahonda	1.500
Domínguez	Viaducto	Villaconejos	Majadahonda	1.300
Valdivieso	Fuencarral	Móstoles	Fuenlabrada	1.500
Barea	*nulo*	*nulo*	Fuenlabrada	5.300

FIGURA 3.34. Resultado de *empleado* ⟖ *trabajo-a-tiempo-completo.*

ción de reunión externa por la izquierda r ⟕ s se puede expresar como:

$$(r \bowtie s) \cup (r - \Pi_R(r \bowtie s)) \times \{(nulo, \ldots, nulo)\}$$

donde la relación constante $\{(nulo, \ldots, nulo)\}$ se encuentra en el esquema $S - R$.

3.3.4. Valores nulos**

En este apartado se define la forma en que las diferentes operaciones del álgebra relacional tratan los valores nulos y las complicaciones que surgen cuando los valores nulos participan en las operaciones aritméticas o en

nombre-empleado	*calle*	*ciudad*	*nombre-sucursal*	*sueldo*
Segura	Tebeo	La Loma	Majadahonda	1.500
Domínguez	Viaducto	Villaconejos	Majadahonda	1.300
Valdivieso	Fuencarral	Móstoles	Fuenlabrada	1.500
Gómez	Bailén	Alcorcón	*nulo*	*nulo*
Barea	*nulo*	*nulo*	Fuenlabrada	5.300

FIGURA 3.35. Resultado de *empleado* ⟗ *trabajo-a-tiempo-completo.*

las comparaciones. Como se verá, a menudo hay varias formas de tratar los valores nulos y, como resultado, las siguientes definiciones pueden ser a veces arbitrarias. Las operaciones y las comparaciones con valores nulos se deberían evitar siempre que sea posible.

Dado que el valor especial *nulo* indica «valor desconocido o no existente», cualquier operación aritmética (como +, –, * y /) que incluya valores nulos debe devolver un valor nulo.

De manera similar, cualquier comparación (como <, <=, >, >= y ≠) que incluya un valor nulo se evalúa al valor especial **desconocido**; no se puede decir si el resultado de la comparación es cierto o falso, así que se dice que el resultado es el nuevo valor lógico *desconocido*.

Las comparaciones que incluyan nulos pueden aparecer dentro de expresiones booleanas que incluyan las operaciones y (conjunción), o (disyunción) y no (negación). Se debe definir la forma en que estas operaciones tratan el valor lógico *desconocido*.

- **y**: (*cierto* **y** *desconocido*) = *desconocido*; (*falso* **y** *desconocido*) = *falso*; (*desconocido* **y** *desconocido*) = *desconocido*.
- **o**: (*cierto* **o** *desconocido*) = *cierto*; (*falso* **o** *desconocido*) = *desconocido*; (*desconocido* **o** *desconocido*) = *desconocido*.
- **no**: (**no** *desconocido*) = *desconocido*.

Ahora es posible describir la forma en que las diferentes operaciones del álgebra relacional tratan los valores nulos. Nuestras definiciones siguen las usadas en el lenguaje SQL.

- **select**: la operación selección evalúa el predicado P en $\sigma_P(E)$ sobre cada tupla de E. Si el predicado devuelve el valor *cierto*, se añade t al resultado. En caso contrario, si el predicado devuelve *desconocido* o *falso*, t no se añade al resultado.
- **reunión**: las reuniones se pueden expresar como un producto cartesiano seguido de una selección. Por tanto, la definición de la forma en que la selección trata los nulos también define la forma en que la operación reunión trata los nulos.

  En una reunión natural $r \bowtie s$ se puede observar de la definición anterior que si dos tuplas, $t_r \in r$ y $t_s \in s$, tienen un valor nulo en un atributo común, entonces las tuplas no casan.
- **proyección**: la operación proyección trata los nulos como cualquier otro valor al eliminar duplicados. Así, si dos tuplas del resultado de la proyección son exactamente iguales, y ambos tienen nulos en los mismos campos, se tratan como duplicados.

  La decisión es un tanto arbitraria porque sin saber cuál es el valor real no se sabe si los dos valores nulos son duplicados o no.
- **unión, intersección, diferencia**: estas operaciones tratan los valores nulos al igual que la operación proyección; tratan las tuplas que tienen los mismos valores en todos los campos como duplicados incluso si algunos de los campos tienen valores nulos en ambas tuplas.

  El comportamiento es un tanto arbitrario, especialmente en el caso de la intersección y la diferencia, dado que no se sabe si los valores reales (si existen) representados por los nulos son los mismos.
- **proyección generalizada**: se describió la manera en que se tratan los nulos en las expresiones al principio del Apartado 3.3.4. Las tuplas duplicadas que contienen valores nulos se tratan como en la operación proyección.

  Cuando hay nulos en los atributos agregados, la operación borra los valores nulos del resultado antes de aplicar la agregación. Si el multiconjunto resultante está vacío, el resultado agregado es nulo.

  Obsérvese que el tratamiento de los valores nulos aquí es diferente que en las expresiones aritméticas ordinarias; se podría haber definido el resultado de una operación de agregación como nulo si incluso sólo uno de los valores agregados es nulo. Sin embargo, esto significaría que un único valor desconocido en un gran grupo podría hacer que el resultado agregado sobre el grupo fuese nulo, y se perdería una gran cantidad de información útil.
- **reunión externa**: las operaciones de reunión externa se comportan como las operaciones reunión, excepto sobre las tuplas que no aparecen en el resultado. Estas tuplas se pueden añadir al resultado (dependiendo de si la operación es $\bowtie$, $\ltimes$ o $\rtimes$) añadiendo nulos.

3.4. MODIFICACIÓN DE LA BASE DE DATOS

Hasta ahora hemos centrado la atención en la extracción de información de la base de datos. En este apartado se abordará la manera de insertar, borrar o modificar información de la base de datos.

Las modificaciones de la base de datos se expresan utilizando la operación asignación. Las asignaciones a las relaciones reales de la base de datos se realizan utilizando la misma notación que se describió para la asignación en el Apartado 3.2.3.

3.4.1. Borrado

Las solicitudes de borrado se expresan básicamente igual que las consultas. Sin embargo, en lugar de mostrar las

tuplas al usuario, se eliminan de la base de datos las tuplas seleccionadas. Sólo se pueden borrar tuplas enteras; no se pueden borrar valores de atributos concretos. En el álgebra relacional los borrados se expresan mediante

$$r \leftarrow r - E$$

donde r es una relación y E es una consulta del álgebra relacional.

He aquí varios ejemplos de solicitudes de borrado del álgebra relacional:

- Borrar todas las cuentas de Gómez.

$$impositor \leftarrow impositor - \sigma_{nombre\text{-}cliente = \text{«Gómez»}}(impositor)$$

- Borrar todos los préstamos con importes entre 0 y 50.

$$préstamo \leftarrow préstamo - \sigma_{importe \geq 0 \text{ and } importe \leq 50}(préstamo)$$

- Borrar todas las cuentas de las sucursales sitas en Getafe.

$$r_1 \leftarrow \sigma_{ciudad\text{-}sucursal = \text{«Getafe»}}(cuenta \bowtie sucursal)$$
$$r_2 \leftarrow \Pi_{nombre\text{-}sucursal,\ número\text{-}cuenta,\ saldo}(r_1)$$
$$cuenta \leftarrow cuenta - r_2$$

  Obsérvese que en el último ejemplo se simplificó la expresión utilizando la asignación a las relaciones temporales (r_1 y r_2).

3.4.2. Inserción

Para insertar datos en una relación hay que especificar la tupla que se va a insertar o escribir una consulta cuyo resultado sea un conjunto de tuplas que vayan a insertarse. Evidentemente, el valor de los atributos de las tuplas insertadas deben ser miembros del dominio de cada atributo. De manera parecida, las tuplas insertadas deben ser de la aridad correcta. En el álgebra relacional las inserciones se expresan mediante

$$r \leftarrow r \cup E$$

donde r es una relación y E es una expresión del álgebra relacional. La inserción de una sola tupla se expresa haciendo que E sea una relación constante que contiene una tupla.

Supóngase que se desea insertar el hecho de que Gómez tiene 1.200 € en la cuenta C-973 en la sucursal de Navacerrada. Hay que escribir

$$cuenta \leftarrow cuenta \cup \{(\text{C-973}, \text{«Navacerrada»}, 1200)\}$$
$$impositor \leftarrow impositor \cup \{(\text{«Gómez»}, \text{C-973})\}$$

De forma más general, puede que se desee insertar tuplas de acuerdo con el resultado de una consulta. Supóngase que se desea ofrecer una nueva cuenta de ahorro con 200 € como regalo a todos los clientes con préstamos concedidos en la sucursal de Navacerrada. Sea el número de préstamo el que se utilice como número de cuenta de esta cuenta de ahorro. Hay que escribir

$$r_1 \leftarrow (\sigma_{nombre\text{-}sucursal = \text{«Navacerrada»}}(prestatario \bowtie préstamo))$$
$$r_2 \leftarrow \Pi_{nombre\text{-}sucursal,\ número\text{-}préstamo}(r_1)$$
$$cuenta \leftarrow cuenta \cup (r_2 \times \{(200)\})$$
$$impositor \leftarrow impositor \cup \Pi_{nombre\text{-}cliente,\ número\text{-}préstamo}(r_1)$$

En lugar de especificar las tuplas como se hizo anteriormente, se especifica un conjunto de tuplas que se insertan en las relaciones *cuenta* e *impositor*. Cada tupla de la relación *cuenta* tiene el *nombre-sucursal* (Navacerrada), un *número-cuenta* (que es igual que el número de préstamo) y el saldo inicial de la nueva cuenta (200 €). Cada tupla de la relación *impositor* tiene como *nombre-cliente* el nombre del prestatario al que se le da la nueva cuenta y el mismo número de cuenta que la correspondiente tupla de *cuenta*.

3.4.3. Actualización

Puede que, en algunas situaciones, se desee modificar un valor de una tupla sin modificar *todos* los valores de la tupla. Se puede utilizar el operador proyección generalizada para realizar esta tarea:

$$r \leftarrow \Pi_{F_1, F_2, \ldots, F_n}(r)$$

donde cada F_i es el i-ésimo atributo de r, si el i-ésimo atributo no está actualizado, o, si hay que actualizar el atributo, una expresión, que sólo implica constantes y los atributos de r, que da el nuevo valor del atributo.

Si se desea seleccionar varias tuplas de r y sólo actualizar esas mismas tuplas, se puede utilizar la expresión siguiente, donde P denota la condición de selección que escoge las tuplas que hay que actualizar:

$$r \leftarrow \Pi_{F_1, F_2, \ldots, F_n}(\sigma_P(r)) \cup (r - \sigma_P(r))$$

Para ilustrar el uso de la operación actualización supóngase que se realiza el pago de los intereses y que hay que aumentar todos los saldos en un 5 por ciento. Hay que escribir

$$cuenta \leftarrow \Pi_{nombre\text{-}sucursal,\ número\text{-}cuenta,\ saldo,\ saldo * 1.05}(cuenta)$$

Supóngase ahora que las cuentas con saldos superiores a 10.000 € reciben un interés del 6 por ciento, mientras que los demás reciben un 5 por ciento. Hay que escribir

$$cuenta \leftarrow \Pi_{NS,\ NC,\ saldo * 1.06}(\sigma_{saldo > 10000}(cuenta)) \cup$$
$$cuenta \leftarrow \Pi_{NS,\ NC,\ saldo * 1.05}(\sigma_{saldo \leq 10000}(cuenta))$$

donde las abreviaturas *NS* y *NC* sustituyen a *nombre-sucursal* y a *número-cuenta*, respectivamente.

3.5. VISTAS

En los ejemplos propuestos hasta ahora se ha operado en el nivel del modelo lógico. Es decir, se ha asumido que el conjunto de relaciones que se da son las relaciones reales guardadas en la base de datos.

No es deseable que todos los usuarios puedan ver la totalidad del modelo lógico. Las consideraciones sobre la seguridad pueden exigir que algunos datos queden ocultos para los usuarios. Considérese una persona que necesita saber el número de préstamo de un cliente pero que no necesita ver el importe del préstamo. Esta persona debería ver una relación descrita en el álgebra relacional mediante

$$\Pi_{\textit{nombre-cliente, número-préstamo, nombre-sucursal}} (\textit{prestatario} \bowtie \textit{préstamo})$$

Aparte de las consideraciones sobre la seguridad puede que se desee crear un conjunto personalizado de relaciones que se adapte mejor que el modelo lógico a la intuición de un usuario concreto. Por ejemplo, puede que un empleado del departamento de publicidad quiera ver una relación que conste de los clientes que tengan abierta una cuenta o concedido un préstamo en el banco y de las sucursales con las que trabajan. La relación que se crearía para ese empleado es

$$\Pi_{\textit{nombre-sucursal, nombre-cliente}} (\textit{impositor} \bowtie \textit{cuenta}) \cup \Pi_{\textit{nombre-sucursal, nombre-cliente}} (\textit{prestatario} \bowtie \textit{préstamo})$$

Las relaciones que no forman parte del modelo lógico pero se hacen visibles a los usuarios como relaciones virtuales se denominan **vistas**. Se puede trabajar con gran número de vistas sobre cualquier conjunto dado de relaciones reales.

3.5.1. Definición de vistas

Las vistas se definen utilizando la instrucción **create view**. Para definir una vista hay que darle un nombre e indicar la consulta que la va a calcular. La forma de la instrucción **create view** es

create view *v* **as** <expresión de consulta>

donde <expresión de consulta> es cualquier expresión legal de consulta del álgebra relacional. El nombre de la vista se representa mediante *v*.

Como ejemplo considérese la vista consistente en las sucursales y sus clientes. Supóngase que se desea que esta vista se denomine *todos-los-clientes*. Esta vista se define de la manera siguiente:

create view *todos-los-clientes* **as**
$\Pi_{\textit{nombre-sucursal, nombre-cliente}} (\textit{impositor} \bowtie \textit{cuenta}) \cup \Pi_{\textit{nombre-sucursal, nombre-cliente}} (\textit{prestatario} \bowtie \textit{préstamo})$

Una vez se ha definido una vista se puede utilizar el nombre de la vista para hacer referencia a la relación virtual que genera la vista. Utilizando la vista *todos-los-clientes* se puede averiguar el nombre de todos los clientes de la sucursal de Navacerrada escribiendo

$$\Pi_{\textit{nombre-cliente}} (\sigma_{\textit{nombre-sucursal}} = \text{«Navacerrada»} (\textit{todos-los-clientes}))$$

Recuérdese que en el Apartado 3.2.1 se escribió la misma consulta sin utilizar vistas.

Los nombres de las vistas pueden aparecer en cualquier lugar en el que pueda encontrarse el nombre de una relación, siempre y cuando no se ejecuten sobre las vistas operaciones de actualización. El asunto de las operaciones de actualización de las vistas se estudia en el Apartado 3.5.2.

La definición de las vistas se diferencia de la operación asignación del álgebra relacional. Supóngase que se define la relación $r1$ de la manera siguiente:

$$r1 \leftarrow \Pi_{\textit{nombre-sucursal, nombre-cliente}} (\textit{impositor} \bowtie \textit{cuenta}) \cup \Pi_{\textit{nombre-sucursal, nombre-cliente}} (\textit{prestatario} \bowtie \textit{préstamo})$$

La operación asignación se evalúa una vez, y $r1$ no cambiará cuando se actualicen las relaciones *impositor*, *cuenta*, *préstamo* o *prestatario*. En cambio, si hay alguna modificación en estas relaciones, el conjunto de tuplas de la vista *todos-los-clientes* también cambia. De manera intuitiva, en cualquier momento dado, el conjunto de tuplas de la relación de vistas se define como el resultado de la evaluación de la expresión de consulta que define en ese momento la vista.

Por tanto, si una relación de vistas se calcula y se guarda, puede quedar desfasada si las relaciones utilizadas para definirla se modifican. En vez de eso, las vistas suelen implementarse de la manera siguiente. Cuando se define una vista, el sistema de la base de datos guarda la definición de la propia vista, en vez del resultado de la evaluación de la expresión del álgebra relacional que la define. Siempre que se utiliza una relación de vistas en una consulta, se sustituye por la expresión de consulta guardada. Por tanto, la relación de vistas vuelve a calcularse siempre que se evalúa la consulta.

Algunos sistemas de bases de datos permiten que se guarden las relaciones de vistas, pero se aseguran de que, si las relaciones reales utilizadas en la definición de la vista cambian, la vista se mantenga actualizada. Estas vistas se denominan **vistas materializadas**. El proceso de mantener actualizada la vista se denomina **mantenimiento de vistas**, que se trata en el Apartado 14.5. Las aplicaciones en las que se utiliza frecuentemente una vista se benefician del uso de vistas materializadas, igual que las aplicaciones que demandan una rápida respuesta a ciertas consultas basadas en las vistas. Las ventajas de

la materialización de una vista para las consultas deben sopesarse frente a los costes de almacenamiento y la sobrecarga añadida por las actualizaciones.

3.5.2. Actualizaciones mediante vistas y valores nulos

Aunque las vistas son una herramienta útil para las consultas, plantean problemas significativos si con ellas se expresan las actualizaciones, las inserciones o los borrados. La dificultad radica en que las modificaciones de la base de datos expresadas en términos de vistas deben traducirse en modificaciones de las relaciones reales en el modelo lógico de la base de datos.

Para ilustrar el problema considérese un empleado que necesita ver todos los datos de préstamos de la relación *préstamo* salvo *importe*. Sea *préstamo-sucursal* la vista dada al empleado. Se define esta vista como

$$\textbf{create view } \textit{préstamo-sucursal} \textbf{ as } \Pi_{\textit{nombre-sucursal, número-préstamo}} (\textit{préstamo})$$

Dado que se permite que los nombres de las vistas aparezcan en cualquier parte en la que estén permitidos los nombres de relaciones, el empleado puede escribir:

$$\textit{préstamo-sucursal} \leftarrow \textit{préstamo-sucursal} \cup \{(\text{P-37}, \text{«Navacerrada»})\}$$

Esta inserción debe representarse mediante una inserción en la relación *préstamo*, dado que *préstamo* es la relación real a partir de la cual se genera la vista *préstamo-sucursal*. Sin embargo, para insertar una tupla en *préstamo* hay que tener algún valor para *importe*. Hay dos enfoques razonables para trabajar con esta inserción:

- Rechazar la inserción y devolver al usuario un mensaje de error.
- Insertar una tupla (P-37, «Navacerrada», *nulo*) en la relación *préstamo*.

Otro problema resultante de la modificación de la base de datos mediante las vistas aparece en una vista como la siguiente:

$$\textbf{create view } \textit{información-crédito} \textbf{ as } \Pi_{\textit{nombre-cliente, importe}}(\textit{prestatario} \bowtie \textit{préstamo})$$

Esta vista da una lista del importe de cada préstamo que tenga concedido cualquier cliente del banco. Considérese la inserción siguiente realizada mediante esta vista:

$$\textit{información-crédito} \leftarrow \textit{información-crédito} \cup \{(\text{«González»}, 1900)\}$$

El único método posible de insertar tuplas en las relaciones *prestatario* y *préstamo* es insertar («González», *nulo*) en *prestatario* y (*nulo*, *nulo*, 1900) en *préstamo*. Así, se obtienen las relaciones mostradas en la Figura 3.36. Sin embargo, esta actualización no tiene el efecto deseado, dado que la relación de vistas *información-crédito* sigue *sin* incluir la tupla («González», 1900). Por tanto, no existe manera de actualizar las relaciones *prestatario* y *préstamo* utilizando valores nulos para obtener la actualización deseada de *información-crédito*.

Debido a este tipo de problemas generalmente no se permiten las modificaciones en las relaciones de vistas excepto en casos limitados. Los diferentes sistemas de bases de datos especifican diferentes condiciones bajo las que se permiten actualizaciones sobre las vistas; véanse los manuales de cada sistema de bases de datos en particular para consultar más detalles. El problema general de la modificación de las bases de datos mediante las vistas ha sido objeto de numerosas investigaciones. Las notas bibliográficas hacen mención de trabajos recientes sobre este asunto.

3.5.3. Vistas definidas utilizando otras vistas

En el Apartado 3.5.1 se mencionó que las relaciones de vistas pueden aparecer en cualquier lugar en que pueda hacerlo el nombre de una relación, salvo las restricciones en el uso de vistas en expresiones para la actualización. Por tanto, se pueden utilizar vistas en la expresión que define otra vista. Por ejemplo, se puede definir la vista *cliente-navacerrada* de la manera siguiente:

$$\textbf{create view } \textit{cliente-navacerrada} \textbf{ as } \Pi_{\textit{nombre-cliente}} (\sigma_{\textit{nombre-sucursal}} = \text{«Navacerrada»} (\textit{todos-los-clientes}))$$

donde *todos-los-clientes* es, a su vez, una relación de vistas.

número-préstamo	*nombre-sucursal*	*importe*
P-11	Collado Mediano	900
P-14	Centro	1.500
P-15	Navacerrada	1.500
P-16	Navacerrada	1.300
P-17	Centro	1.000
P-23	Moralzarzal	2.000
P-93	Becerril	500
nulo	*nulo*	1.900

nombre-cliente	*número-préstamo*
Fernández	P-16
Gómez	P-23
Gómez	P-11
López	P-15
Pérez	P-93
Santos	P-17
Sotoca	P-14
Valdivieso	P-17
González	nulo

FIGURA 3.36. Tuplas insertadas en *préstamo* y en *prestatario*.

La **expansión de vistas** es una manera de definir el significado de las vistas definidas en términos de otras vistas. El procedimiento asume que las definiciones de vistas no son **recursivas**; es decir, ninguna vista se usa en su propia definición, bien directa o indirectamente a través de otras definiciones de vistas. Por ejemplo, si v_1 se usa en la definición de v_2, se usa en la definición de v_3, y v_3 se usa en la definición de v_1, entonces v_1, v_2 y v_3 son recursivas. Las definiciones de vistas recursivas son útiles en algunos casos, y se volverá a ellas en el contexto del lenguaje Datalog, en el Apartado 5.2.

Sea la vista v_1 definida mediante una expresión e_1 que puede contener a su vez relaciones de vistas. Las relaciones de vistas representan a las expresiones que definen las vistas y, por tanto, se pueden sustituir por las expresiones que las definen. Si se modifica una expresión sustituyendo una relación de vistas por su definición, la expresión resultante puede seguir conteniendo otras relaciones de vistas. Por tanto, la expansión de vistas de una expresión repite la etapa de sustitución de la manera siguiente:

repeat
  Buscar todas las relaciones de vistas v_i de e_1
  Sustituir la relación de vistas v_i por la expresión que define v_i
until no queden más relaciones de vistas en e_1

Mientras las definiciones de las vistas no sean recursivas el bucle concluirá. Por tanto, una expresión e que contenga relaciones de vistas puede entenderse como la expresión resultante de la expansión de vistas de e, que no contiene ninguna relación de vistas.

Como ilustración de la expansión de vistas considérese la expresión siguiente:

$$\sigma_{nombre\text{-}cliente = \text{«Martín»}} (cliente\text{-}navacerrada)$$

El procedimiento de expansión de vistas produce inicialmente

$$\sigma_{nombre\text{-}cliente = \text{«Martín»}} (\Pi_{nombre\text{-}cliente} (\sigma_{nombre\text{-}sucursal = \text{«Navacerrada»}} (todos\text{-}los\text{-}clientes)))$$

luego produce

$$\sigma_{nombre\text{-}cliente = \text{«Martín»}} (\Pi_{nombre\text{-}cliente} (\sigma_{nombre\text{-}sucursal = \text{«Navacerrada»}} (\Pi_{nombre\text{-}sucursal,\ nombre\text{-}cliente} (impositor \bowtie cuenta) \cup \Pi_{nombre\text{-}sucursal,\ nombre\text{-}cliente} (prestatario \bowtie préstamo))))$$

No hay más usos de las relaciones de vistas y concluye la expansión de vistas.

3.6. EL CÁLCULO RELACIONAL DE TUPLAS

Cuando escribimos una expresión del álgebra relacional proporcionamos una serie de procedimientos que generan la respuesta a la consulta. El cálculo relacional de tuplas, en cambio, es un lenguaje de consulta **no procedimental**. Describe la información deseada sin dar un procedimiento específico para obtenerla.

Las consultas se expresan en el cálculo relacional de tuplas como

$$\{t \mid P(t)\}$$

es decir, son el conjunto de todas las tuplas tales que el predicado P es cierto para t. Siguiendo la notación utilizada previamente, se utiliza $t[A]$ para denotar el valor de la tupla t en el atributo A y $t \in r$ para denotar que la tupla t está en la relación r.

Antes de dar una definición formal del cálculo relacional de tuplas se volverán a examinar algunas de las consultas para las que se escribieron expresiones de álgebra relacional en el Apartado 3.2.

3.6.1. Consultas de ejemplo

Supóngase que se desea averiguar *nombre-sucursal*, *número-préstamo* e *importe* de los préstamos superiores a 1.200 €:

$$\{t \mid t \in préstamo \wedge t[importe] > 1200\}$$

Supóngase que sólo se desea obtener el atributo *número-préstamo*, en vez de todos los atributos de la relación *préstamo*. Para escribir esta consulta en el cálculo relacional de tuplas hay que escribir una expresión para una relación del esquema (*número-préstamo*). Se necesitan las tuplas de (*número-préstamo*) tales que hay una tupla en *préstamo* con el atributo *importe* > 1200. Para expresar esta solicitud hay que utilizar el constructor «existe» de la lógica matemática. La notación

$$\exists\, t \in r\, (Q\,(t))$$

significa «existe una tupla t en la relación r tal que el predicado $Q(t)$ es verdadero».

Utilizando esta notación se puede escribir la consulta «Averiguar el número de préstamo de todos los préstamos por importe superior a 1.200 €» como

$$\{t \mid \exists\, s \in préstamo\ (t[número\text{-}préstamo] = s[número\text{-}préstamo] \wedge s[importe] > 1200)\}$$

En español la expresión anterior se lee «el conjunto de todas las tuplas t tales que existe una tupla s en la relación *préstamo* para la que los valores de t y de s para el

atributo *número-préstamo* son iguales y el valor de *s* para el atributo *importe* es mayor que 1.200 €».

La variable tupla *t* sólo se define para el atributo *número-préstamo*, dado que es el único atributo para el que se especifica una condición para *t*. Por tanto, el resultado es una relación de (*número-préstamo*).

Considérese la consulta «Averiguar el nombre de todos los clientes que tienen concedido un préstamo en la sucursal de Navacerrada». Esta consulta es un poco más compleja que las anteriores, dado que implica a dos relaciones: *prestatario* y *préstamo*. Como se verá, sin embargo, todo lo que necesita es que tengamos dos instrucciones «existe» en la expresión de cálculo relacional de tuplas, relacionadas mediante *y* ($\wedge$). La consulta se escribe de la manera siguiente:

$$\{t \mid \exists\, s \in \textit{prestatario}\ (t[\textit{número-préstamo}] = s[\textit{número-préstamo}] \wedge \exists\, u \in \textit{préstamo}\ (u[\textit{número-préstamo}] = s[\textit{número-préstamo}] \wedge u[\textit{nombre-sucursal}] = \text{«Navacerrada»}))\}$$

En español esta expresión es «el conjunto de todas las tuplas (*nombre-cliente*) para las que el cliente tiene un préstamo concedido en la sucursal de Navacerrada». La variable tupla *u* asegura que el cliente es prestatario de la sucursal de Navacerrada. La variable tupla *s* está restringida para que corresponda al mismo número de préstamo que *s*. El resultado de esta consulta se muestra en la Figura 3.37.

Para averiguar todos los clientes del banco que tienen concedido un préstamo, una cuenta abierta, o ambas cosas, se utilizó la operación unión del álgebra relacional. En el cálculo relacional de tuplas harán falta dos instrucciones «existe» relacionadas por *o* ($\vee$):

$$\{t \mid \exists\, s \in \textit{prestatario}\ (t[\textit{nombre-cliente}] = s[\textit{nombre-cliente}]) \vee \exists\, u \in \textit{impositor}\ (t[\textit{nombre-cliente}] = u[\textit{nombre-cliente}])\}$$

Esta expresión da el conjunto de todas las tuplas de *nombre-cliente* tales que se cumple al menos una de las condiciones siguientes:

- *nombre-cliente* aparece en alguna tupla de la relación *prestatario* como prestatario del banco.
- *nombre-cliente* aparece en alguna tupla de la relación *impositor* como impositor del banco.

Si algún cliente tiene concedido un préstamo y una cuenta abierta en el banco, ese cliente sólo aparece una vez en el resultado, ya que la definición matemática de conjunto no permite elementos duplicados. El resultado de esta consulta se mostró previamente en la Figura 3.12.

nombre-cliente
Fernández
López

FIGURA 3.37. Nombre de todos los clientes que tienen concedido un préstamo en la sucursal de Navacerrada.

Si *sólo* queremos conocer los clientes que tienen en el banco una cuenta *y* un préstamo, todo lo que hay que hacer es cambiar en la expresión anterior la *o* ($\vee$) por una *y* ($\wedge$).

$$\{t \mid \exists\, s \in \textit{prestatario}\ (t[\textit{nombre-cliente}] = s[\textit{nombre-cliente}]) \wedge \exists\, u \in \textit{impositor}\ (t[\textit{nombre-cliente}] = u[\textit{nombre-cliente}])\}$$

El resultado de esta consulta se mostró en la Figura 3.20.

Considérese ahora la consulta «Averiguar todos los clientes que tienen una cuenta abierta en el banco pero no tienen concedido ningún préstamo». La expresión del cálculo relacional de tuplas para esta consulta es parecida a las expresiones que se acaban de ver, salvo el uso del símbolo *no* ($\neg$):

$$\{t \mid \exists\, u \in \textit{impositor}\ (t[\textit{nombre-cliente}] = u[\textit{nombre-cliente}]) \wedge \neg\, \exists\, s \in \textit{prestatario}\ (t[\textit{nombre-cliente}] = s[\textit{nombre-cliente}])\}$$

La expresión del cálculo relacional de tuplas anterior utiliza la instrucción $\exists\, u \in \textit{impositor}\ (\ldots)$ para exigir que el cliente tenga una cuenta abierta en el banco, y utiliza la instrucción $\neg\, \exists\, s \in \textit{prestatario}\ (\ldots)$ para borrar a aquellos clientes que aparecen en alguna tupla de la relación *prestatario* por tener un préstamo del banco. El resultado de esta consulta apareció en la Figura 3.13.

La consulta que se tomará ahora en consideración utiliza la implicación, denotada por $\Rightarrow$. La fórmula $P \Rightarrow Q$ es lógicamente equivalente a $\neg\, P \vee Q$. El uso de la implicación en lugar de *no* y *o* suele sugerir una interpretación más intuitiva de la consulta en español.

Considérese la consulta que se utilizó en el Apartado 3.2.3 para ilustrar la operación división: «Averiguar todos los clientes que tienen una cuenta en todas las sucursales sitas en Arganzuela». Para escribir esta consulta en el cálculo relacional de tuplas se introduce el constructor «para todo», denotado por $\forall$. La notación

$$\forall\, t \in r\ (Q\ (t))$$

significa «Q es verdadera para todas las tuplas *t* de la relación *r*».

La expresión para la consulta se escribe de la manera siguiente:

$$\{t \mid \exists\, r \in \textit{cliente}\ (r[\textit{nombre-cliente}] = t[\textit{nombre-cliente}] \wedge (\forall\, u \in \textit{sucursal}\ (u[\textit{ciudad-sucursal}] = \text{«Arganzuela»} \Rightarrow \exists\, s \in \textit{impositor}\ (t[\textit{nombre-cliente}] = s[\textit{nombre-cliente}] \wedge \exists\, w \in \textit{cuenta}\ (w[\textit{número-cuenta}] = s[\textit{número-cuenta}] \wedge w[\textit{nombre-sucursal}] = u[\textit{nombre-sucursal}])))))\}$$

En español esta expresión se interpreta como «el conjunto de todos los clientes (es decir, las tuplas t (*nombre-cliente*)) tales que, para todas las tuplas u de la relación *sucursal*, si el valor de u en el atributo *ciudad-sucursal* es Arganzuela, el cliente tiene una cuenta en la sucursal cuyo nombre aparece en el atributo *nombre-sucursal* de u».

Nótese que hay una sutileza en la consulta anterior: si no hay ninguna sucursal en Arganzuela, todos los nombres de cliente satisfacen la condición. La primera línea de la expresión de consulta es crítica en este caso: sin la condición

$$\exists\, r \in \textit{cliente}\ (r[\textit{nombre-cliente}] = t[\textit{nombre-cliente}]$$

si no hay sucursal en Arganzuela, cualquier valor de t (incluyendo los valores que no son nombres de cliente en la relación *cliente*) valdría.

3.6.2. Definición formal

Ahora se tiene la preparación necesaria para una definición formal. Las expresiones del cálculo relacional de tuplas son de la forma

$$\{t \mid P(t)\}$$

donde P es una *fórmula*. En una fórmula pueden aparecer varias variables tupla. Se dice que una variable tupla es una *variable libre* a menos que esté cuantificada mediante $\exists$ o $\forall$. Por tanto, en

$$t \in \textit{préstamo} \wedge \exists\, s \in \textit{cliente}\ (t[\textit{nombre-sucursal}] = s[\textit{nombre-sucursal}])$$

t es una variable libre. La variable tupla s se denomina variable *ligada*.

Las fórmulas de cálculo relacional de tuplas se construyen con *átomos*. Los átomos tienen una de las formas siguientes:

- $s \in r$, donde s es una variable tupla y r es una relación (no se permite el uso del operador $\notin$)
- $s[x]\ \Theta\ u[y]$, donde s y u son variables tuplas, x es un atributo en el que está definida s, y es un atributo en el que está definida u y Θ es un operador de comparación ($<, \leq, =, \neq, >, \geq$); es necesario que los atributos x e y tengan dominios cuyos miembros puedan compararse mediante Θ
- $s[x]\ \Theta\ c$, donde s es una variable tupla, x es un atributo en el que está definida s, Θ es un operador de comparación y c es una constante en el dominio del atributo x

Las fórmulas se construyen a partir de los átomos utilizando las reglas siguientes:

- Un átomo es una fórmula.
- Si P_1 es una fórmula, también lo son $\neg P_1$ y (P_1).
- Si P_1 y P_2 son fórmulas, también lo son $P_1 \vee P_2$, $P_1 \wedge P_2$ y $P_1 \Rightarrow P_2$.
- Si $P_1(s)$ es una fórmula que contiene una variable tupla libre s, y r es una relación,

$$\exists\, s \in r\ (P_1(s)) \quad \text{y} \quad \forall\, s \in r\ (P_1(s))$$

también son fórmulas

Igual que en el álgebra relacional, se pueden escribir expresiones equivalentes que no sean idénticas en apariencia. En el cálculo relacional de tuplas estas equivalencias incluyen las tres reglas siguientes:

1. $P_1 \wedge P_2$ es equivalente a $\neg\,(\neg\,(P_1) \vee \neg\,(P_2))$.
2. $\forall\, t \in r\ (P_1(t))$ es equivalente a $\neg\, \exists\, t \in r\ (\neg\, P_1(t))$.
3. $P_1 \Rightarrow P_2$ es equivalente a $\neg\,(P_1) \vee P_2$.

3.6.3. Seguridad de las expresiones

Queda un último asunto por tratar. Las expresiones del cálculo relacional de tuplas pueden generar relaciones infinitas. Supóngase que se escribió la expresión

$$\{t \mid \neg\,(t \in \textit{préstamo})\}$$

Hay infinitas tuplas que no están en *préstamo*. La mayor parte de estas tuplas contienen valores que ni siquiera aparecen en la base de datos. Resulta evidente que no se desea permitir ese tipo de expresiones.

Para ayudar a definir las restricciones del cálculo relacional de tuplas se introduce el concepto de **dominio** de una fórmula relacional de tuplas, P. De manera intuitiva, el dominio de P, denotado por $dom(P)$, es el conjunto de todos los valores a los que P hace referencia. Esto incluye a los valores mencionados en la propia P, así como a los valores que aparezcan explícitamente en P o en una o en varias relaciones cuyos nombres aparezcan en P. Así, el dominio de P es el conjunto de todos los valores que aparecen explícitamente en una o más relación cuyos nombres aparecen en P. Por ejemplo, $dom(t \in \textit{préstamo} \wedge t[\textit{importe}] > 1200)$ es el conjunto que contiene a 1200 y el conjunto de todos los valores que aparecen en *préstamo*. Además, $dom(\neg\,(t \in \textit{préstamo}))$ es el conjunto de todos los valores que aparecen en *préstamo*, dado que la relación *préstamo* se menciona en la expresión.

Se dice que una expresión $\{t \mid P(t)\}$ es *segura* si todos los valores que aparecen en el resultado son valores de $dom(P)$. La expresión $\{t \mid \neg\,(t \in \textit{préstamo})\}$ no es segura. Obsérvese que $dom(\neg\,(t \in \textit{préstamo}))$ es el conjunto de todos los valores que aparecen en *préstamo*. Sin embargo, es posible tener una tupla t que no esté en *préstamo* que contenga valores que no aparezcan en *préstamo*. El resto de ejemplos de expresiones del cálculo relacional de tuplas que se han escrito en este apartado son seguros.

3.6.4. Potencia expresiva de los lenguajes

El cálculo relacional de tuplas restringido a expresiones seguras es equivalente en potencia expresiva al álgebra relacional básica (con los operadores $\cup$, $-$, $\times$, σ y ρ, pero sin los operadores relacionales extendidos tales como la proyección generalizada G y las operaciones de reunión externa). Por tanto, para cada expresión del álgebra relacional hay una expresión equivalente del cálculo relacional de tuplas, y para cada expresión del cálculo relacional de tuplas hay una expresión equivalente del álgebra relacional. No se probará aquí esta afirmación; las notas bibliográficas contienen referencias a la demostración. Algunas partes de la misma se incluyen en los ejercicios. El cálculo relacional de tuplas no tiene ningún equivalente de la operación agregación, pero se puede extender para contenerla. La extensión del cálculo relacional de tuplas para manejar las expresiones aritméticas es sencilla.

3.7. EL CÁLCULO RELACIONAL DE DOMINIOS**

Hay una segunda forma de cálculo relacional denominada **cálculo relacional de dominios**. Esta forma utiliza variables de *dominio* que toman sus valores del dominio de un atributo, en vez de tomarlos de una tupla completa. El cálculo relacional de dominios, sin embargo, se halla estrechamente relacionado con el cálculo relacional de tuplas.

3.7.1. Definición formal

Las expresiones del cálculo relacional de dominios son de la forma

$$\{< x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

donde $x_1, x_2, \ldots, x_n$ representan las variables de dominio, P representa una fórmula compuesta de átomos, como era el caso en el cálculo relacional de tuplas. Los átomos del cálculo relacional de dominios tienen una de las formas siguientes:

- $< x_1, x_2, \ldots, x_n > \in r$, donde r es una relación con n atributos y $x_1, x_2, \ldots, x_n$ son variables de dominio o constantes de dominio.
- $x \; \Theta \; y$, donde x e y son variables de dominio y Θ es un operador de comparación ($<, \leq, =, \neq, >, \geq$). Se exige que los atributos x e y tengan dominios que puedan compararse mediante Θ.
- $x \; \Theta \; c$, donde x es una variable de dominio, Θ es un operador de comparación y c es una constante del dominio del atributo para el que x es una variable de dominio.

Las fórmulas se construyen a partir de los átomos utilizando las reglas siguientes:

- Un átomo es una fórmula.
- Si P_1 es una fórmula, también lo son $\neg P_1$ y (P_1).
- Si P_1 y P_2 son fórmulas, también lo son $P_1 \vee P_2$, $P_1 \wedge P_2$ y $P_1 \Rightarrow P_2$.
- Si $P_1(x)$ es una fórmula en x, donde x es una variable de dominio,

  $$\exists \, x < (P_1 (x)) \quad \text{y} \quad \forall \, x \, (P_1 (x))$$

  también son fórmulas

Como notación abreviada se escribe

$$\exists \, a, b, c \, (P(a, b, c))$$

en lugar de

$$\exists \, a \, (\exists \, b \, (\exists \, c \, (P(a, b, c))))$$

3.7.2. Consultas de ejemplo

Ahora se van a aportar consultas del cálculo relacional de dominios para los ejemplos considerados anteriormente. Obsérvese la similitud de estas expresiones con las expresiones correspondientes del cálculo relacional de tuplas

- Averiguar el nombre de la sucursal, el número de préstamo y el importe de los préstamos superiores a 1.200 €:

  $$\{< p, s, i > \mid < p, s, i > \in \textit{préstamo} \wedge i > 1200\}$$

- Averiguar todos los números de préstamo de los préstamos por importe superior a 1.200 €:

  $$\{< p > \mid \exists \, s, i \, (< p, s, i > \in \textit{préstamo} \wedge i > 1200)\}$$

Aunque la segunda consulta tenga un aspecto muy parecido al de la que se escribió para el cálculo relacional de tuplas, hay una diferencia importante. En el cálculo de tuplas, cuando se escribe $\exists \, s$ para alguna variable tupla s, se vincula inmediatamente con una relación escribiendo $\exists \, s \in r$. Sin embargo, cuando se escribe $\exists \, s$ en el cálculo de dominios, s no se refiere a una tupla, sino a un valor de dominio. Por tanto, el dominio de la variable s no está restringido hasta que la subfórmula $p, s, i \in \textit{préstamo}$ restringe s a los nom-

bres de sucursal que aparecen en la relación *préstamo*. Por ejemplo:

- Averiguar el nombre de todos los clientes que tienen concedido un préstamo en la sucursal de Navacerrada y averiguar el importe del préstamo:

  $\{< n, c > \mid \exists\, l\, (< n, p > \in \textit{prestatario} \wedge \exists\, s\, (< p, s, i > \in \textit{préstamo} \wedge s = \text{«Navacerrada»}))\}$

- Averiguar el nombre de todos los clientes que tienen concedido un préstamo, una cuenta abierta, o ambas cosas, en la sucursal de Navacerrada:

  $\{< n > \mid \exists\, p\, (< n, p > \in \textit{prestatario} \wedge \exists\, s, i\ (< p, s, i > \in \textit{préstamo} \wedge s = \text{«Navacerrada»})) \vee \exists\, c\, (< n, c > \in \textit{impositor} \wedge \exists\, s, i\ (< c, s, i > \in \textit{cuenta} \wedge s = \text{«Navacerrada»}))\}$

- Averiguar el nombre de todos los clientes que tienen una cuenta abierta en todas las sucursales sitas en Arganzuela:

  $\{< c > \mid \exists\, s, t\, (< c, s, t > \in \textit{cliente}) \wedge \forall\, x, y, z\ (< x, y, z > \in \textit{sucursal}) \wedge y = \text{«Arganzuela»} \Rightarrow \exists\, a, b\, (< x, a, b > \in \textit{cuenta} \wedge (< c, a > \in \textit{impositor}))\}$

  En español la expresión anterior se interpreta como «el conjunto de todas las tuplas *c* (*nombre-cliente*) tales que, para todas las tuplas *x*, *y*, *z* (*nombre-sucursal*, *ciudad-sucursal*, *activos*), si la ciudad de la sucursal es Arganzuela, las siguientes afirmaciones son verdaderas»:

  — Existe una tupla de la relación *cuenta* con número de cuenta *a* y nombre de sucursal *x*

  — Existe una tupla de la relación *impositor* con cliente *c* y número de cuenta *a*

3.7.3. Seguridad de las expresiones

Ya se observó que en el cálculo relacional de tuplas es posible escribir expresiones que pueden generar relaciones infinitas. Esto llevó a definir la *seguridad* de las expresiones de cálculo relacional de tuplas. Se produce una situación parecida en el cálculo relacional de dominios. Las expresiones como

$$\{< p, s, i > \mid \neg\, (< p, s, i > \in \textit{préstamo})\}$$

no son seguras porque permiten valores del resultado que no están en el dominio de la expresión.

En el cálculo relacional de dominios también hay que tener en cuenta la forma de las fórmulas dentro de las instrucciones «existe» y «para todo». Considérese la expresión

$$\{< x > \mid \exists\, y\, (< x, y \in r) \wedge \exists\, z\, (\neg\, (< x, z > \in r) \wedge P(x, z))\}$$

donde *P* es una fórmula que implica a *x* y a *z*. Se puede probar la primera parte de la fórmula, $\exists\, y\, (< x, y \in r)$, tomando en consideración sólo los valores de *r*. Sin embargo, para probar la segunda parte de la fórmula, $\exists\, z\, (\neg\, (< x, z > \in r) \wedge P(x, z))$, hay que tomar en consideración valores de *z* que no aparecen en *r*. Dado que todas las relaciones son finitas, no aparece en *r* un número infinito de valores. Por tanto, no resulta posible en general probar la segunda parte de la fórmula $\exists\, z\, (\neg\, (< x, z > \in r) \wedge P(x, z))$, hay que tomar en consideración valores de *z* que no aparecen en *r*. Dado que todas las relaciones son finitas, no aparece en *r* un número infinito de valores. Por tanto, no es posible en general probar la segunda parte de la fórmula sin tomar en consideración un número infinito de valores de *z*. En vez de eso, se añaden restricciones para prohibir expresiones como la anterior.

En el cálculo relacional de tuplas se restringió cualquier variable cuantificada existencialmente a variar sobre una relación concreta. Dado que no se hizo así en el cálculo de dominios, hay que añadir reglas a la definición de seguridad para tratar los casos parecidos a los del ejemplo. Se dice que la expresión

$$\{< x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

es segura si se cumplen todas las condiciones siguientes:

1. Todos los valores que aparecen en las tuplas de la expresión son valores de $dom(P)$.
2. Para cada subfórmula «existe» de la forma $\exists\, x\, (P_1(x))$, la subfórmula es cierta si y sólo si hay un valor *x* en $dom(P_1)$ tal que $P_1(x)$ es verdadero.
3. Para cada subfórmula «para todo» de la forma $\forall\, x\, (P_1(x))$, la subfórmula es verdadera si y sólo si $P_1(x)$ es verdadero para todos los valores *x* de $dom(P_1)$.

El propósito de las reglas adicionales es asegurar que se puedan probar las subfórmulas «para todo» y «existe» sin tener que probar infinitas posibilidades. Considérese la segunda regla de la definición de seguridad. Para que $\exists\, x\, (P_1(x))$ sea verdadero sólo hay que encontrar una *x* para la que $P_1\, (x)$ lo sea. En general, habría que probar infinitos valores. Sin embargo, si la expresión es segura, se sabe que se puede restringir la atención a los valores de $dom(P_1)$. Esta restricción reduce las tuplas que hay que tomar en consideración a un número finito.

La situación de las subfórmulas de la forma $\forall\, x\, (P_1(x))$ es parecida. Para asegurar que $\forall\, x\, (P_1(x))$ es verdadero hay que probar en general todos los valores posibles, por lo que hay que examinar infinitos valores. Como antes, si se sabe que la expresión es segura, basta con probar $P_1(x)$ para los valores tomados de $dom(P_1)$.

Todas las expresiones del cálculo relacional de dominios que se han incluido en las consultas de ejemplo de este apartado son seguras.

3.7.4. Potencia expresiva de los lenguajes

Cuando el cálculo relacional de dominios se restringe a expresiones seguras es equivalente en potencia expresiva al cálculo relacional de tuplas restringido a expresiones seguras. Dado que se observó anteriormente que el cálculo relacional de tuplas restringido es equivalente al álgebra relacional, los tres lenguajes siguientes son equivalentes:

- El álgebra relacional básica (sin las operaciones extendidas)
- El cálculo relacional de tuplas restringido a expresiones seguras
- El cálculo relacional de dominios restringido a expresiones seguras

El cálculo relacional de dominios tampoco tiene equivalente para la operación agregación, pero se puede extender para contenerla, y su extensión para el tratamiento de expresiones aritméticas es sencilla.

3.8. RESUMEN

- El **modelo de datos relacional** se basa en un conjunto de tablas. El usuario del sistema de bases de datos puede consultar esas tablas, insertar nuevas tuplas, borrar tuplas y actualizar (modificar) las tuplas. Hay varios lenguajes para expresar estas operaciones.
- El **álgebra relacional** define un conjunto de operaciones algebraicas que operan sobre tablas y devuelven tablas como resultado. Estas operaciones se pueden combinar para obtener expresiones que expresan las consultas deseadas. El álgebra define las operaciones básicas usadas en los lenguajes de consulta relacionales.
- Las operaciones del álgebra relacional se pueden dividir en :
  - — Operaciones básicas
  - — Operaciones adicionales que se pueden expresar en términos de las operaciones básicas
  - — Operaciones extendidas, algunas de las cuales añaden mayor poder expresivo al álgebra relacional
- Las bases de datos se pueden modificar con la **inserción**, el **borrado** y la **actualización** de tuplas. Se usó el álgebra relacional con el **operador de asignación** para expresar estas modificaciones.
- Los diferentes usuarios de una base de datos compartida pueden aprovecharse de vistas individualizadas de la base de datos. Las vistas son «relaciones virtuales» definidas mediante expresiones de consulta.
- Las vistas son mecanismos útiles para simplificar las consultas a la base de datos, pero la modificación de la base de datos mediante las vistas puede tener consecuencias potencialmente desventajosas. Por tanto, los sistemas de bases de datos restringen estrictamente las actualizaciones mediante vistas.
- Por razones de eficiencia del procesamiento de las consultas, una vista puede estar **materializada**, es decir, la consulta se evalúa y el resultado se almacena físicamente. Cuando las relaciones de la base de datos se actualizan, la vista materializada se debe actualizar correspondientemente.
- El **cálculo relacional de tuplas** y el **cálculo relacional de dominios** son lenguajes no procedimentales que representan la potencia básica necesaria en un lenguaje de consultas relacionales. El álgebra relacional básica es un lenguaje procedimental que es equivalente en potencia a ambas formas del cálculo relacional cuando se restringen a las expresiones seguras.
- El álgebra relacional y los cálculos relacionales son lenguajes rígidos, formales, que no resultan adecuados para los usuarios ocasionales de los sistemas de bases de datos. Los sistemas comerciales de bases de datos, por tanto, utilizan lenguajes con más «azúcar sintáctico». En los Capítulos 4 y 5 se tomarán en consideración los tres lenguajes comerciales más influyentes: **SQL**, que está basado en el álgebra relacional, **QBE** y **Datalog**, que están basados en el cálculo relacional de dominios.

TÉRMINOS DE REPASO

- Agrupación
- Álgebra relacional
- Cálculo relacional de dominios
- Cálculo relacional de tuplas
- Clave externa
  - Relación referenciada
  - Relación referenciante
- Claves
- Definición de vistas
- Dominio atómico
- Diagrama de esquema
- Ejemplar de la base de datos
- Ejemplar de la relación
- Esquema de la base de datos
- Esquema de la relación
- Expansión de vistas
- Lenguaje de consulta
- Lenguaje procedimental
- Lenguaje no procedimental
- Modificación de la base de datos
  - Actualización
  - Borrado
  - Inserción
- Multiconjuntos
- Operaciones adicionales
  - División /
  - Intersección de conjuntos $\cap$
  - Reunión natural $\bowtie$
- Operaciones del álgebra relacional
  - Diferencia de conjuntos –
  - Producto cartesiano $\times$
  - Proyección Π
  - Renombramiento ρ
  - Selección σ
  - Unión $\cup$
- Operaciones del álgebra relacional extendida
  - Agregación G
  - Proyección generalizada Π
  - Reunión externa
    - Reunión externa completa $\mathbin{⟗}$
    - Reunión externa por la derecha $\mathbin{⟖}$
    - Reunión externa por la izquierda $\mathbin{⟕}$
- Operación asignación
- Potencia expresiva de los lenguajes
- Relación
- Seguridad de las expresiones
- Tabla
- Valor nulo
- Valores nulos
- Variable tupla
- Vistas
- Vistas recursivas

EJERCICIOS

3.1. Diséñese una base de datos relacional para la oficina de registro de una universidad. La oficina conserva datos sobre cada curso, incluyendo el profesor, el número de estudiantes matriculados y la hora y el lugar de las clases. Por cada pareja estudiante-curso se guarda una calificación.

3.2. Descríbanse las diferencias de significado entre los términos *relación* y *esquema de la relación*. Ilústrese la respuesta haciendo referencia a la solución propuesta para el Ejercicio 3.1.

3.3. Diséñese una base de datos relacional correspondiente al diagrama E-R de la Figura 3.38.

3.4. En el Capítulo 2 se mostró la manera de representar los conjuntos de relaciones de varios a varios, de varios a uno, de uno a varios y de uno a uno. Explíquese la manera en que las claves primarias ayudan a representar estos conjuntos de relaciones en el modelo relacional.

3.5. Considérese la base de datos relacional de la Figura 3.39. Dese una expresión del álgebra relacional, otra del cálculo relacional de tuplas y una tercera del cálculo relacional de dominios para cada una de las consultas siguientes:

a. Averiguar los nombres de todos los empleados que trabajan para el Banco Importante.

b. Averiguar el nombre y la ciudad de residencia de todos los empleados que trabajan para el Banco Importante.

c. Averiguar el nombre, la calle y la ciudad de residencia de todos los empleados que trabajan para el Banco Importante y ganan más de 2.000.000 de pesetas anuales.

d. Averiguar el nombre de todos los empleados de esta base de datos que viven en la misma ciudad que la compañía para la que trabajan.

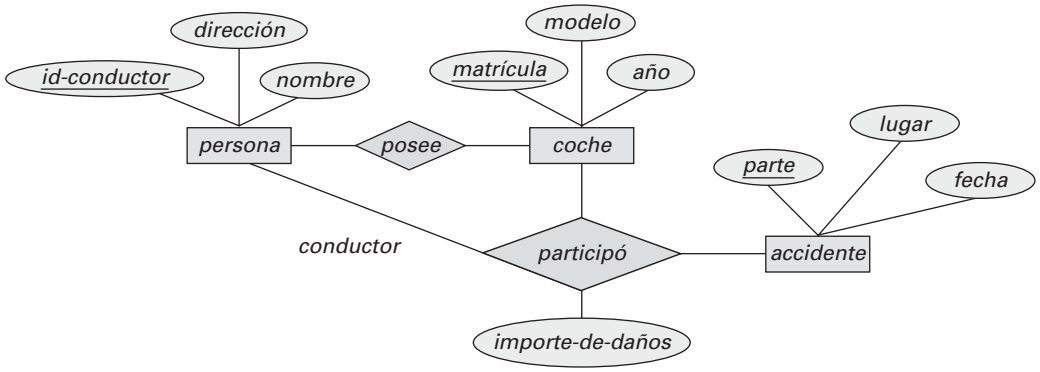

FIGURA 3.38. Diagrama E-R.

e. Averiguar el nombre de todos los empleados que viven en la misma ciudad y en la misma calle que sus jefes.

f. Averiguar el nombre de todos los empleados de esta base de datos que no trabajan para el Banco Importante.

g. Averiguar el nombre de todos los empleados que ganan más que cualquier empleado del Banco Pequeño.

h. Supóngase que las compañías pueden estar instaladas en ciudades pequeñas. Hállense todas las compañías instaladas en cada ciudad en la que está instalado el Banco Pequeño.

3.6. Considérese la relación de la Figura 3.21, que muestra el resultado de la consulta «Averígüese el nombre de todos los clientes que tienen concedido un préstamo en el banco». Vuélvase a escribir la consulta para incluir no sólo el nombre, sino también la ciudad de residencia de cada cliente. Obsérvese que ahora el cliente Sotoca ya no aparece en el resultado, aunque en realidad tiene un préstamo concedido por el banco.

a. Explíquese el motivo de que Sotoca no aparezca en el resultado.

b. Supóngase que se desea que Sotoca aparezca en el resultado. ¿Cómo habría que modificar la base de datos para conseguirlo?

c. Una vez más, supóngase que se desea que Sotoca aparezca en el resultado. Escríbase una consulta utilizando una reunión externa que cumpla esta condición sin que haya que modificar la base de datos.

3.7. Las operaciones de reunión externa amplían la operación reunión natural de manera que las tuplas de las relaciones participantes no se pierdan en el resultado de la reunión. Descríbase la manera en que la operación reunión zeta puede ampliarse para que las tuplas de la relación de la izquierda, las de la relación de la derecha o las de ambas relaciones no se pierdan en el resultado de una reunión zeta.

empleado (<u>*nombre-empleado*</u>, *calle*, *ciudad*)
trabaja (<u>*nombre-empleado*</u>, *nombre-empresa*, *sueldo*)
empresa (<u>*nombre-empresa*</u>, *ciudad*)
jefe (<u>*nombre-empleado*</u>, *nombre-jefe*)

FIGURA 3.39. Base de datos relacional para los Ejercicios 3.5 y 3.10.

3.8. Considérese la base de datos regional de la Figura 3.39. Dese una expresión del álgebra relacional para cada petición:

a. Modificar la base de datos de manera que Santos viva ahora en Tres Cantos.

b. Dar a todos los empleados del Banco Importante un aumento de sueldo del 10 por ciento.

c. Dar a todos los jefes de la base de datos un aumento de sueldo del 10 por ciento.

d. Dar a todos los jefes de la base de datos un aumento de sueldo del 10 por ciento, a menos que el sueldo resultante sea mayor que 100.000 €. En este caso, dar sólo un aumento del 3 por ciento.

e. Borrar todas las tuplas de los empleados de Banco Pequeño de la relación *trabajo*.

3.9. Utilizando el ejemplo bancario, escríbanse consultas del álgebra relacional para averiguar las cuentas abiertas por más de dos clientes:

a. utilizando una función de agregación.

b. sin utilizar funciones de agregación.

3.10. Considérese la base de datos relacional de la Figura 3.38. Dese una expresión del álgebra relacional para cada una de las consultas siguientes:

a. Averiguar la compañía con mayor número de empleados.

b. Averiguar la compañía con la nómina (suma de sueldos de sus empleados) más reducida.

c. Averiguar las compañías cuyos empleados ganen un sueldo más elevado, en media, que el sueldo medio del Banco Importante.

3.11. Dense dos motivos por los que se puede decidir definir una vista.

3.12. Cítense dos problemas importantes del procesamiento de la operación actualización expresadas en términos de vistas.

3.13. Sean los siguientes esquemas de relaciones:

$$R = (A, B, C)$$
$$S = (D, E, F)$$

Sean las relaciones $r(R)$ y $s(S)$. Dese una expresión del cálculo relacional de tuplas que sea equivalente a cada una de las expresiones siguientes:

a. $\Pi_A(R)$
b. $\sigma_{B=17}(r)$
c. $r \times s$
d. $\Pi_{A,F}(\sigma_{C=D}(r \times s))$

3.14. Sea $R = (A, B, C)$ y sean r_1 y r_2 relaciones del esquema R. Dese una expresión del cálculo relacional de dominios que sea equivalente a las expresiones siguientes:

a. $\Pi_A(r_1)$
b. $\sigma_{B=17}(r_1)$
c. $r_1 \cup r_2$
d. $r_1 \cap r_2$
e. $r_1 - r_2$
f. $\Pi_{A,B}(r_1) \bowtie \Pi_{B,C}(r_2)$

3.15. Repítase el Ejercicio 3.5 usando el cálculo relacional de tuplas y el de dominios.

3.16. Sean $R = (A, B)$ y $S = (A, C)$ y sean $r(R)$ y $s(S)$ relaciones. Escríbanse expresiones del álgebra relacional equivalentes a las expresiones siguientes del cálculo relacional de dominios:

a. $\{< a > \mid \exists\, b\, (< a, b > \in r \wedge b = 17)\}$
b. $\{< a, b, c > \mid < a, b > \in r \wedge < a, c > \in s\}$
c. $\{< a > \mid \exists\, b\, (< a, b > \in r) \vee \forall\, c\, (\exists\, d\, (< d, c > \in s) \Rightarrow < a, c > \in s)\}$
d. $\{< a > \mid \exists\, c\, (< a, c > \in s) \wedge \exists\, b_1, b_2\, (< a, b_1 > \in r \wedge < c, b_2 > \in r \wedge b_1 > b_2))\}$

3.17. Sea $R = (A, B)$ y $S = (A, C)$ y sean $r(R)$ y $s(S)$ relaciones. Utilizando la constante especial *nulo*, escríbanse expresiones del cálculo relacional de tuplas equivalentes a cada una de las expresiones siguientes:

a. $r \bowtie\!\!\!\!\subset s$
b. $r \rhd\!\!\!\!\subset s$
c. $r \rhd\!\!\!\bowtie s$

3.18. Dense dos motivos por los que se puedan introducir valores nulos en la base de datos.

3.19. Algunos sistemas permiten los valores nulos *marcados*. Un valor nulo marcado $\perp_i$ es igual a sí mismo, pero si $i \neq j$, $\perp_i \neq \perp_j$. Una aplicación de valores nulos marcados debe permitir ciertas actualizaciones mediante el uso de vistas. Considérese la vista *información-crédito* (Apartado 3.5). Muéstrese la manera en que se pueden utilizar los valores nulos marcados para permitir la inserción de la tupla («González», 1900) mediante *información-crédito*.

NOTAS BIBLIOGRÁFICAS

El modelo relacional fue propuesto por E. F. Codd del Laboratorio de investigación de San José de IBM a finales de los años sesenta [Codd, 1970]. Este trabajo motivó la concesión a Codd del prestigioso Premio Turing de la ACM en 1981 (Codd [1982]).

Siguiendo el trabajo original de Codd se constituyeron varios proyectos de investigación con el objetivo de crear sistemas de bases de datos relacionales prácticos, incluyendo System R del Laboratorio de investigación de San José de IBM, Ingres en la Universidad de California en Berkeley, Query-by-Example en el Centro de investigación T. J. Watson de IBM (*IBM T. J. Watson Research Center*) y el vehículo de prueba relacional (*Peterlee Relational Test Vehicle*, PRTV) del Centro científico de IBM (*IBM Scientific Center*) en Peterlee, Reino Unido. System R se discute en Astrahan et al. [1976], Astrahan et al. [1979] y en Chamberlin et al. [1981]. Ingres se discute en Stonebraker [1980], Stonebraker [1986b] y en Stonebraker et al. [1976]. Query-by-Example se describe en Zloof [1977]. PRTV se describe en Todd [1976].

Actualmente están disponibles comercialmente numerosos productos de bases de datos relacionales. Ejemplos de ello son DB2 de IBM, Ingres, Oracle, Sybase, Informix y Microsoft SQL Server. Ejemplos de productos de bases de datos para las computadoras personales son Microsoft Access, dBase y FoxPro. La información sobre estos productos puede hallarse en sus manuales respectivos.

En la mayor parte de los textos sobre bases de datos se incluye una discusión general del modelo relacional de datos. Atzeni y De Antonellis [1993] y Maier [1983] son textos dedicados exclusivamente al modelo relacional de datos. La definición original del álgebra relacional está en Codd [1970]; la del cálculo relacional de tuplas en Codd [1972]. En Codd [1972] se encuentra una prueba formal de la equivalencia del cálculo relacional de tuplas y del álgebra relacional.

Se han propuesto varias ampliaciones del cálculo relacional. Klug [1982] y Escobar-Molano et al. [1993] describen ampliaciones para funciones de agregación escalares. En Codd [1979] se presentan ampliaciones del modelo relacional y discusiones sobre la incorporación de los valores nulos al álgebra relacional (el modelo RM/T), así como las la de las reuniones externas. Codd [1990] es un compendio de los trabajos de E. F. Codd sobre el modelo relacional. Las reuniones externas también se discuten en Date [1993b]. El problema de la actualización de las bases de datos relacionales mediante vistas se aborda en Bancilhon y Spyratos [1981], Cosmadakis y Papadimitriou [1984], Dayal y Bernstein [1978, 1982] y Langerak [1990]. El Apartado 14.5 trata el mantenimiento de las vistas materializadas, y las referencias a la literatura sobre ello se pueden encontrar al final de ese capítulo.

PARTE

II

BASES DE DATOS RELACIONALES

Una base de datos relacional es un repositorio compartido de datos. Para hacer disponibles los datos de un base de datos relacional a los usuarios hay que considerar varios aspectos. Uno es la forma en que los usuarios solicitan los datos: ¿cuáles son los diferentes lenguajes de consulta que usan? El Capítulo 4 trata el lenguaje SQL, que es el lenguaje de consulta más ampliamente usado actualmente. El Capítulo 5 trata otros dos lenguajes de consulta, QBE y Datalog, que ofrecen enfoques alternativos a la consulta de datos relacionales.

Otro aspecto es la integridad de datos y la seguridad; las bases de datos necesitan proteger los datos del daño provocado por los usuarios, ya sean intencionados o no. El componente de mantenimiento de la integridad de una base de datos asegura que las actualizaciones no violan las restricciones de integridad que hayan especificado sobre los datos. El componente de seguridad de una base de datos incluye la autenticación de usuarios y el control de acceso para restringir las posibles acciones de cada usuario. El Capítulo 6 trata los aspectos de integridad y seguridad. Estos aspectos se presentan independientemente del modelo de datos, pero se estudian en el contexto del modelo de datos relacional para ejemplificarlos. Las restricciones de integridad forman la base del diseño de bases de datos relacionales, que se estudian en el Capítulo 7.

El diseño de bases de datos relacionales —el diseño del esquema relacional— es el primer paso en la construcción de aplicaciones de bases de datos. El diseño de esquemas se trató informalmente en capítulos anteriores. Sin embargo, hay principios que se pueden usar para distinguir los buenos diseños de bases de datos. Se formalizan mediante varias «formas normales», que ofrecen diferentes compromisos entre la posibilidad de inconsistencias y la eficiencia de ciertas consultas. El Capítulo 7 describe el diseño formal de esquemas relacionales.

CAPÍTULO

4 SQL

Los lenguajes formales descritos en el Capítulo 3 proporcionan una notación concisa para la representación de consultas. Sin embargo, los sistemas de bases de datos comerciales necesitan un lenguaje de consultas cómodo para el usuario. En este capítulo se estudia el lenguaje comercial de mayor influencia, SQL. SQL usa una combinación de álgebra relacional y construcciones del cálculo relacional.

Aunque el lenguaje SQL se considere un lenguaje de consultas, contiene muchas otras capacidades además de la consulta en bases de datos. Incluye características para definir la estructura de los datos, para la modificación de los datos en la base de datos y para la especificación de restricciones de seguridad.

No se pretende proporcionar un manual de usuario completo para SQL. Por el contrario, se presentan las construcciones y conceptos fundamentales de SQL. Las distintas implementaciones de SQL pueden diferenciarse en detalles, o pueden admitir sólo un subconjunto del lenguaje completo.

4.1. INTRODUCCIÓN

IBM desarrolló la versión original en su Laboratorio de Investigación de San José (*San José Research Center*, actualmente Centro de Investigación de Almadén, *Almadén Research Center*). IBM implementó el lenguaje, originalmente denominado Sequel, como parte del proyecto System R, a principios de 1970. El lenguaje Sequel ha evolucionado desde entonces y su nombre ha pasado a ser SQL (*Structured Query Language*, Lenguaje estructurado de consultas). Actualmente, numerosos productos son compatibles con el lenguaje SQL. SQL se ha establecido como *el* lenguaje estándar de bases de datos relacionales.

En 1986, ANSI (*American National Standards Institute*, Instituto Nacional Americano de Normalización) e ISO (*International Standards Organization*, Organización Internacional de Normalización), publicaron una norma SQL, denominada SQL-86. En 1987, IBM publicó su propia norma de SQL corporativo, Interfaz de bases de datos para arquitecturas de aplicación a sistemas (*Systems Application Architecture Database Interface*, SAA-SQL). En 1989 se publicó una norma extendida para SQL denominada SQL-89 y actualmente los sistemas de bases de datos son normalmente compatibles al menos con las características de SQL-89. La siguiente versión de la norma fue SQL-92 y la versión más reciente es SQL:1999. Las notas bibliográficas proporcionan referencias a esas normas.

En este apartado se presenta una visión general de SQL basada en la norma SQL-92 ampliamente implementada. La norma SQL:1999 es un superconjunto de la norma SQL-92; en este capítulo se tratan algunas características de SQL:1999 y se proporciona un estudio más detallado en el Capítulo 9. Muchos sistemas de bases de datos soportan algunas de las nuevas constructoras de SQL:1999, aunque ningún sistema de bases datos actual soporta todas las nuevas constructoras. También hay ser consciente de que algunos sistemas de bases de datos ni siquiera soportan todas las características de SQL-92 y de que muchas bases de datos proporcionan características no estándar que no se tratan aquí.

El lenguaje SQL tiene varios componentes:

- **Lenguaje de definición de datos** (LDD). El LDD de SQL proporciona órdenes para la definición de esquemas de relación, borrado de relaciones, creación de índices y modificación de esquemas de relación.
- **Lenguaje interactivo de manipulación de datos** (LMD). El LMD de SQL incluye un lenguaje de consultas, basado tanto en el álgebra relacional como en el cálculo relacional de tuplas. Incluye también órdenes para insertar, borrar y modificar tuplas de la base de datos.
- **Definición de vistas.** El LDD de SQL incluye órdenes para la definición de vistas.
- **Control de transacciones.** SQL incluye órdenes para la especificación del comienzo y final de transacciones.
- **SQL incorporado** y **SQL dinámico.** SQL dinámico e incorporado define cómo se pueden incorporar las instrucciones SQL en lenguajes de pro-

gramación de propósito general, tales como C, C++, Java, PL/I, Cobol, Pascal y Fortran.

- **Integridad.** El LDD de SQL incluye órdenes para la especificación de las restricciones de integridad que deben satisfacer los datos almacenados en la base de datos. Las actualizaciones que violen las restricciones de integridad se rechazan.
- **Autorización.** El LDD de SQL incluye órdenes para especificar derechos de acceso para las relaciones y vistas.

En este capítulo se estudia el LMD y las características básicas del LDD de SQL. También se describe brevemente SQL incorporado y dinámico, incluyendo las normas ODBC y JDBC para la interacción con una base de datos desde programas escritos en lenguajes C y Java. Las características de SQL que dan soporte a la integridad y autorización se describen en el Capítulo 6, mientras que el Capítulo 9 esboza las extensiones orientadas a objeto de SQL.

Los ejemplos de este capítulo y posteriores se basarán en una empresa bancaria, con los siguientes esquemas de relación:

Esquema-sucursal = (*nombre-sucursal*, *ciudad-sucursal*, *activo*)
Esquema-cliente = (*nombre-cliente*, *calle-cliente*, *ciudad-cliente*)
Esquema-préstamo = (*número-préstamo*, *nombre-sucursal*, *importe*)
Esquema-prestatario = (*nombre-cliente*, *número-préstamo*)
Esquema-cuenta = (*número-cuenta*, *nombre-sucursal*, *saldo*)
Esquema-impositor = (*nombre-cliente*, *número-cuenta*)

Nótese que en este capítulo, como en el resto del texto, se usan nombres separados por guiones para los esquemas, relaciones y atributos para facilitar su lectura. Sin embargo, en los sistemas SQL actuales, los guiones no son partes válidas de un nombre (se tratan como el operador menos). Una forma simple de traducir los nombres que se usan aquí a nombres SQL válidos es reemplazar todos los guiones por el símbolo de subrayado («_»). Por ejemplo, se usa *nombre_sucursal* en lugar de *nombre-sucursal*.

4.2. ESTRUCTURA BÁSICA

Una base de datos relacional consiste en un conjunto de relaciones, a cada una de las cuales se le asigna un nombre único. Cada relación tiene una estructura similar a la presentada en el Capítulo 3. SQL permite el uso de valores nulos para indicar que el valor o bien es desconocido, o no existe. Se fijan criterios que permiten al usuario especificar a qué atributos no se puede asignar valor nulo, como estudiaremos en el Apartado 4.11.

La estructura básica de una expresión SQL consiste en tres cláusulas: **select**, **from** y **where**.

- La cláusula **select** corresponde a la operación proyección del álgebra relacional. Se usa para listar los atributos deseados del resultado de una consulta.
- La cláusula **from** corresponde a la operación producto cartesiano del álgebra relacional. Lista las relaciones que deben ser analizadas en la evaluación de la expresión.
- La cláusula **where** corresponde al predicado selección del álgebra relacional. Es un predicado que engloba a los atributos de las relaciones que aparecen en la cláusula **from**.

Un hecho histórico desafortunado es que el término *select* tiene un significado diferente en SQL que en el álgebra relacional. A continuación se resaltan las diferentes interpretaciones, a fin de minimizar la posible confusión.

Una consulta típica en SQL tiene la forma

select $A_1, A_2, \ldots, A_n$
from $r_1, r_2, \ldots, r_m$
where P

Cada *Ai* representa un atributo, y cada *ri* una relación. *P* es un predicado. La consulta es equivalente a la expresión del álgebra relacional

$$\Pi\, A_1, A_2, \ldots, A_n\, (\sigma_P\ (r_1 \times r_2 \times \ldots \times r_m))$$

Si se omite la cláusula **where**, el predicado P es **cierto**. Sin embargo, con diferencia a la expresión del álgebra relacional, el resultado de la consulta SQL puede contener varias copias de algunas tuplas; este aspecto se analizará de nuevo en el Apartado 4.2.8.

SQL forma el producto cartesiano de las relaciones incluidas en la cláusula **from**, lleva a cabo la selección del álgebra relacional usando el predicado de la cláusula **where** y entonces proyecta el resultado sobre los atributos de la cláusula **select**. En la práctica, SQL puede convertir la expresión en una forma equivalente que puede ser procesada más eficientemente. Las cuestiones relativas a la eficiencia se analizan en los Capítulos 13 y 14.

4.2.1. La cláusula select

El resultado de una consulta SQL es, por supuesto, una relación. Considérese una consulta simple, usando el

ejemplo bancario, «Obtener los números de todas las sucursales en la relación *préstamo*»:

select *nombre-sucursal*
from *préstamo*

El resultado es una relación consistente en el único atributo *nombre-sucursal*.

Los lenguajes formales de consulta están basados en la noción matemática de que una relación es un conjunto. Así, nunca aparecen tuplas duplicadas en las relaciones. En la práctica, la eliminación de duplicados consume tiempo. Sin embargo, SQL (como la mayoría de los lenguajes de consulta comerciales) permite duplicados en las relaciones, así como en el resultado de las expresiones SQL. Así, la consulta anterior listará cada *nombre-sucursal* una vez por cada tupla en la que aparece en la relación *préstamo*.

En aquellos casos donde se quiera forzar la eliminación de duplicados, se insertará la palabra clave **distinct** después de **select**. Por lo tanto, se puede reescribir la consulta anterior como

select distinct *nombre-sucursal*
from *préstamo*

si se desean eliminar los duplicados.

Es importante resaltar que SQL permite usar la palabra clave **all** para especificar explícitamente que no se eliminan los duplicados:

select all *nombre-sucursal*
from *préstamo*

Como de manera predeterminada se realiza la retención de duplicados, de ahora en adelante no se usará la palabra clave **all** en los ejemplos. Para asegurar la eliminación de duplicados en el resultado de los ejemplos de consultas, se usará la cláusula **distinct** siempre que sea necesario. En la mayoría de las consultas donde no se utiliza **distinct**, el número exacto de copias duplicadas de cada tupla que resultan de la consulta no es importante. Sin embargo, el número es importante en ciertas aplicaciones; este aspecto se volverá a tratar en el Apartado 4.2.8.

El símbolo asterisco «*» se puede usar para denotar «todos los atributos». Así, el uso de *préstamo*.* en la cláusula **select** anterior indicaría que todos los atributos de préstamo serían seleccionados. Una cláusula **select** de la forma **select** * indica que se deben seleccionar todos los atributos de todas las relaciones que aparecen en la cláusula **from**.

La cláusula **select** puede contener también expresiones aritméticas que contengan los operadores, +, −, ∗ y / operando sobre constantes o atributos de la tuplas. Por ejemplo, la consulta

select *nombre-sucursal*, *número-préstamo*,
importe * 100
from *préstamo*

devolverá una relación que es igual que la relación *préstamo*, salvo que el atributo *importe* está multiplicado por 100.

SQL también proporciona tipos de datos especiales, tales como varias formas del tipo *fecha* y permite varias funciones aritméticas para operar sobre esos tipos.

4.2.2. La cláusula where

A continuación se ilustra con un ejemplo el uso de la cláusula **where** en SQL. Considérese la consulta «Obtener todos los números de préstamo para préstamos hechos en la sucursal con nombre Navacerrada, en los que el importe sea superior a 1.200 €». Esta consulta puede escribirse en SQL como

select *número-préstamo*
from *préstamo*
where *nombre-sucursal* = 'Navacerrada' **and**
importe > 1200

SQL usa las conectivas lógicas **and**, **or** y **not** (en lugar de los símbolos matemáticos ∧, ∨ y ¬) en la cláusula **where**. Los operandos de las conectivas lógicas pueden ser expresiones que contengan los operadores de comparación <, <=, >, >=, = y <>. SQL permite usar los operadores de comparación para comparar cadenas y expresiones aritméticas, así como tipos especiales, tales como el tipo fecha.

SQL incluye un operador de comparación **between** para simplificar las cláusulas **where** que especifica que un valor sea menor o igual que un valor y mayor o igual que otro valor. Si se desea obtener el número de préstamo de aquellos préstamos por importes entre 90.000 € y 100.000 €, se puede usar la comparación **between** para escribir

select *número-préstamo*
from *préstamo*
where *importe* **between** 90000 **and** 100000

en lugar de

select *número-préstamo*
from *préstamo*
where *importe* <= 100000 **and** *importe* >= 90000

De forma análoga, se puede usar el operador de comparación **not between**.

4.2.3. La cláusula from

Finalmente, se estudia el uso de la cláusula **from**. La cláusula **from** define por sí misma un producto cartesiano de las relaciones que aparecen en la cláusula. Escribir una expresión SQL para la reunión natural es una tarea relativamente fácil, puesto que la reunión natu-

ral se define en términos de un producto cartesiano, una selección y una proyección.

La expresión del álgebra relacional se escribe como sigue:

$$\Pi_{\textit{nombre-cliente, número-préstamo,importe}}(\textit{prestatario} \bowtie \textit{préstamo})$$

para la consulta «Para todos los clientes que tienen un préstamo en el banco, obtener los nombres, números de préstamo e importes». Esta consulta puede escribirse en SQL como

select *nombre-cliente*, *prestatario.número-préstamo*, *importe*
from *prestatario*, *préstamo*
where *prestatario.número-préstamo* = *préstamo.número-préstamo*

Nótese que SQL usa la notación *nombre-relación.nombre-atributo*, como lo hace el álgebra relacional, para evitar ambigüedad en los casos en que un atributo aparece en el esquema de más de una relación. También se podría haber escrito *prestatario.nombre-cliente* en lugar de *nombre-cliente*, en la cláusula **select**. Sin embargo, como el atributo *nombre-cliente* aparece sólo en una de las relaciones de la cláusula **from**, no existe ambigüedad al escribir *nombre-cliente*.

Se puede extender la consulta anterior y considerar un caso más complicado en el que se pide además qué clientes poseen un préstamo en la sucursal Navacerrada: «Obtener los nombres, números de préstamo e importes de todos los clientes que tienen un préstamo en la sucursal Navacerrada». Para escribir esta consulta será necesario establecer dos restricciones en la cláusula **where**, relacionadas con la conectiva lógica **and**:

select *nombre-cliente*, *prestatario.número-préstamo*, *importe*
from *prestatario*, *préstamo*
where *prestatario.número-préstamo* = *préstamo.número-préstamo* **and** *nombre-sucursal*= 'Navacerrada'

SQL-92 incluye extensiones para llevar a cabo reuniones naturales y reuniones externas en la cláusula **from**. Esto se estudiará en el Apartado 4.10.

4.2.4. La operación renombramiento

SQL proporciona un mecanismo para renombrar tanto relaciones como atributos. Para ello utiliza la cláusula **as**, que tiene la forma siguiente:

nombre-antiguo ***as*** *nombre-nuevo*

la cláusula **as** puede aparecer tanto en **select** como en **from**.

Considérese de nuevo la consulta anterior:

select distinct *nombre-cliente*, *prestatario.número-préstamo*, *importe*
from *prestatario*, *préstamo*
where *prestatario.número-préstamo* = *préstamo.número-préstamo*

El resultado de esta consulta es una relación con los atributos siguientes:

nombre-cliente, *número-préstamo*, *importe*.

Los nombres de los atributos en el resultado se derivan de los nombres de los atributos de la relación que aparece en la cláusula **from**.

Sin embargo, no se pueden derivar siempre los nombres de este modo. En primer lugar, dos relaciones que aparecen en la cláusula **from** pueden tener atributos con el mismo nombre, en cuyo caso, un nombre de atributo se duplica en el resultado. En segundo lugar, si se incluye una expresión aritmética en la cláusula **select**, los atributos resultantes no tienen el mismo nombre. Y en tercer lugar, incluso si un nombre de atributo se puede derivar de las relaciones base, como en el ejemplo anterior, se puede querer cambiar el nombre del atributo en el resultado. Para todo ello, SQL proporciona una forma de renombrar los atributos de una relación resultado.

Por ejemplo, si se quisiera renombrar el atributo *número-préstamo*, asociándole el nombre de *id-préstamo*, se podría reescribir la consulta anterior del siguiente modo

select *nombre-cliente*, *prestatario.número-préstamo* **as** *id-préstamo, importe*
from *prestatario*, *préstamo*
where *prestatario.número-préstamo* = *préstamo.número-préstamo*

4.2.5. Variables tupla

La cláusula **as** es particularmente útil en la definición del concepto de variables tupla, como se hace en el cálculo relacional de tuplas. Una variable tupla en SQL se debe asociar con una relación concreta. Las variables tupla se definen en la cláusula **from** mediante el uso de la cláusula **as**. Como ejemplo, a continuación se reescribe la consulta «Obtener los nombres y números de préstamo de todos los clientes que tienen un préstamo en el banco» como sigue

select *nombre-cliente*, *T.número-préstamo*, *S.importe*
from *prestatario* **as** *T*, *préstamo* **as** *S*
where *T.número-préstamo* = *S.número-préstamo*

Nótese que se define la variable tupla en la cláusula **from**, colocándola después del nombre de la relación a la cual está asociada y detrás de la palabra clave **as** (la palabra clave **as** es opcional). Al escribir expresiones

de la forma *nombre-relación.nombre-atributo*, el nombre de la relación es, en efecto, una variable tupla definida implícitamente.

Las variables tupla son de gran utilidad para comparar dos tuplas de la misma relación. Hay que recordar que, en los casos de este tipo, se puede usar la operación renombramiento del álgebra relacional. Si se desea formular la consulta «Obtener los nombres de todas las sucursales que poseen un activo mayor que al menos una sucursal situada en Barcelona», se puede escribir la siguiente expresión SQL

select distinct *T.nombre-sucursal*
from *sucursal* **as** *T*, *sucursal* **as** *S*
where *T.activo* > *S.activo* **and** *S.ciudad-sucursal*
= 'Barcelona'

Obsérvese que no se puede utilizar la notación *sucursal.activo*, puesto que no estaría claro a qué aparición de *sucursal* se refiere.

SQL permite usar la notación $(v_1, v_2, \ldots, v_n)$ para designar una tupla de aridad n que contiene los valores v_1, v_2, …, v_n. Los operadores de comparación se pueden utilizar sobre tuplas, y el orden se define lexicográficamente. Por ejemplo $(a_1, a_2) \leq (b_1, b_2)$ es cierto si $(a_1 < b_1)$ o si se cumple que $(a_1 = b_1) \wedge (a_2 \leq b_2)$; análogamente, dos tuplas son iguales si lo son todos sus atributos.

4.2.6. Operaciones sobre cadenas

SQL especifica las cadenas encerrándolas entre comillas simple, como 'Navacerrada', como se vio anteriormente. Un carácter comilla que sea parte de una cadena se puede especificar usando dos caracteres comilla; por ejemplo, la cadena «El carácter ' se puede ver en esta cadena» se puede especificar como 'El carácter '' se puede ver en esta cadena'.

La operación más usada sobre cadenas es el encaje de patrones, para el que se usa el operador **like**. Para la descripción de patrones se utilizan los dos caracteres especiales siguientes:

- Tanto por ciento (%): El carácter % encaja con cualquier subcadena.
- Subrayado (_): El carácter _ encaja con cualquier carácter.

Los patrones son muy sensibles, esto es, los caracteres en mayúsculas no encajan con los caracteres en minúscula, o viceversa. Para ilustrar el encaje de patrones, considérense los siguientes ejemplos:

- 'Nava%' encaja con cualquier cadena que empiece con «Nava».
- '%cer%' encaja con cualquier cadena que contenga «cer» como subcadena, por ejemplo 'Navacerrada', 'Cáceres' y 'Becerril'.
- '_ _ _' encaja con cualquier cadena de tres caracteres.
- '_ _ _%' encaja con cualquier cadena de al menos tres caracteres.

Los patrones se expresan en SQL utilizando el operador de comparación **like**. Considérese la consulta siguiente: «Obtener los nombres de todos los clientes cuyas calles contengan la subcadena 'Mayor'». Esta consulta se podría escribir como sigue

select *nombre-cliente*
from *cliente*
where *calle-cliente* **like** '%Mayor%'

Para que los patrones puedan contener los caracteres especiales patrón (esto es, % y _), SQL permite la especificación de un carácter de escape. El carácter de escape se utiliza inmediatamente antes de un carácter especial patrón para indicar que ese carácter especial va a ser tratado como un carácter normal. El carácter de escape para una comparación **like** se define utilizando la palabra clave **escape**. Para ilustrar esto, considérense los siguientes patrones, los cuales utilizan una barra invertida (\) como carácter de escape:

- **like** 'ab\%cd%' **escape** '\' encaja con todas las cadenas que empiecen por ab%cd .
- **like** 'ab\\cd%' **escape** '\' encaja con todas las cadenas que empiecen por ab\cd .

SQL permite buscar discordancias en lugar de concordancias utilizando el operador de comparación **not like**.

SQL también proporciona una variedad de funciones que operan sobre cadenas de caracteres, tales como la concatenación (usando «||»), la extracción de subcadenas, el cálculo de la longitud de las cadenas, la conversión a mayúsculas y minúsculas, etc. SQL:1999 también ofrece una operación **similar to** que proporciona un encaje de patrones más potente que la operación **like**; la sintaxis para especificar patrones es similar a la usada en Unix para expresiones regulares.

4.2.7. Orden en la presentación de las tuplas

SQL ofrece al usuario cierto control sobre el orden en el cual se presentan las tuplas de una relación. La cláusula **order by** hace que las tuplas resultantes de una consulta se presenten en un cierto orden. Para listar en orden alfabético todos los clientes que tienen un préstamo en la sucursal Navacerrada se escribirá:

select distinct *nombre_cliente*
from *prestatario*, *préstamo*
where *prestatario.número-préstamo*
= *préstamo.número-préstamo* **and**
nombre-sucursal = 'Navacerrada'
order by *nombre-cliente*

De manera predeterminada la cláusula **order by** lista los elementos en orden ascendente. Para especificar el tipo de ordenación se puede incluir la cláusula **desc** para orden descendente o **asc** para orden ascendente. Además, se puede ordenar con respecto a más de un atributo. Si se desea listar la relación *préstamo* en orden descendente para *importe*. Si varios préstamos tienen el mismo importe, se ordenan ascendentemente según el número de préstamo. Esta consulta en SQL se escribe del modo siguiente:

select *
from *préstamo*
order by *importe* **desc**, *número-préstamo* **asc**

Para ejecutar una consulta que contiene la cláusula **order by**, SQL tiene que llevar a cabo una ordenación. Como ordenar un gran número de tuplas puede ser costoso, es conveniente ordenar sólo cuando sea estrictamente necesario.

4.2.8. Duplicados

La utilización de relaciones con duplicados se ha mostrado útil en diversas situaciones. SQL no sólo define formalmente las tuplas que están en el resultado de una consulta, sino también el número de copias de cada una de esas tuplas que aparece en el resultado. La semántica de duplicados de una consulta SQL se puede definir utilizando versiones de los operadores relacionales para *multiconjuntos*. A continuación se definen las versiones multiconjunto de varios de los operadores del álgebra relacional. Dadas las relaciones multiconjunto r_1 y r_2,

1. Si existen c_1 copias de la tupla t_1 en r_1, y t_1 satisface la selección σ_θ, entonces hay c_1 copias de t_1 en $\sigma_\theta(r_1)$.
2. Para cada copia de la tupla t_1 en r_1, hay una copia de la tupla $\Pi_A(t_1)$ en $\Pi_A(r_1)$, donde $\Pi_A(t_1)$ denota la proyección de la tupla única t_1.
3. Si existen c_1 copias de la tupla t_1 en r_1 y c_2 copias de la tupla t_2 en r_2, entonces hay $c_1 * c_2$ copias de la tupla $t_1.t_2$ en $r_1 \times r_2$.

Por ejemplo, supóngase que las relaciones r_1 con esquema (A, B) y r_2 con esquema (C) son los multiconjuntos siguientes:

$$r_1 = \{(1,a), (2,a)\} \qquad r_2 = \{(2), (3), (3)\}$$

Entonces, $\Pi_B(r_1)$ sería $\{(a), (a)\}$, mientras que $\Pi_B(r_1) \times r_2$ sería

$$\{(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)\}$$

Se puede ahora definir cuántas copias de cada tupla aparecen en el resultado de una consulta SQL. Una consulta SQL de la forma

select $A_1, A_2, \ldots, A_n$
from $r_1, r_2, \ldots, r_m$
where P

es equivalente a la expresión del álgebra relacional

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

usando las versiones multiconjunto de los operadores relacionales σ, Π y $\times$.

4.3. OPERACIONES SOBRE CONJUNTOS

Las operaciones de SQL-92 **union**, **intersect** y **except** operan sobre relaciones y corresponden a las operaciones del álgebra relacional $\cup$, $\cap$ y $-$. Al igual que la unión, intersección y diferencia de conjuntos en el álgebra relacional, las relaciones que participan en las operaciones han de ser *compatibles*; esto es, deben tener el mismo conjunto de atributos.

A continuación se demuestra cómo se pueden formular en SQL varias de las consultas de ejemplo consideradas en el Capítulo 3 utilizando consultas que incluyen las operaciones **union**, **intersect** y **except** de dos conjuntos. Los dos conjuntos utilizados serán: el conjuntos de todos los clientes que tienen una cuenta en el banco, que puede obtenerse con:

select *nombre-cliente*
from *impositor*

y el conjunto de todos los clientes que tienen un préstamo en el banco, que puede obtenerse con:

select *nombre-cliente*
from *prestatario*

A partir de ahora, las letras *i* y *p* se utilizarán para hacer referencia a las relaciones obtenidas como resultado de las dos consultas anteriores.

4.3.1. La operacion unión

Para encontrar todos los clientes que poseen un préstamo, una cuenta o las dos cosas en el banco, se escribirá:

(**select** *nombre-cliente*
from *impositor*)

```
union
(select nombre-cliente
 from prestatario)
```

A diferencia de la cláusula **select**, la operación **union** (unión) elimina duplicados automáticamente. Así, en la consulta anterior, si un cliente —por ejemplo, Santos— tiene varias cuentas o préstamos (o ambas cosas) en el banco, entonces Santos aparecerá sólo una vez en el resultado.

Para conservar los duplicados, se utilizará **union all** en lugar de **union**:

```
(select nombre-cliente
 from impositor)
union all
(select nombre-cliente
 from prestatario)
```

El número de tuplas duplicadas en el resultado es igual al número total de duplicados que aparecen en *i* y *p*. Así, si Santos tuviese tres cuentas y dos préstamos en el banco, entonces en el resultado aparecerían cinco tuplas con el nombre de Santos.

4.3.2. La operación intersección

Para encontrar todos los clientes que tienen tanto un préstamo como una cuenta en el banco, se escribirá:

```
(select distinct nombre-cliente
 from impositor)
intersect
(select distinct nombre-cliente
 from prestatario)
```

La operacion **intersect** (intersección) elimina duplicados automáticamente. Así, en la consulta anterior, si un cliente —por ejemplo, Santos— tiene varias cuentas o préstamos (o ambas cosas) en el banco, entonces Santos aparecerá solo una vez en el resultado.

Para conservar los duplicados se utilizará **intersect all** en lugar de **intersect**:

```
(select nombre-cliente
 from impositor)
intersect all
(select nombre-cliente
 from prestatario)
```

El número de tuplas duplicadas en el resultado es igual al mínimo número de duplicados que aparecen en *i* y *p*. Así, si Santos tuviese tres cuentas y dos préstamos en el banco, entonces en el resultado de la consulta aparecerían dos tuplas con el nombre de Santos.

4.3.3. La operación excepto

Para encontrar todos los clientes que tienen cuenta pero no tienen ningún préstamo en el banco se escribirá:

```
(select distinct nombre-cliente
 from impositor)
except
(select distinct nombre-cliente
 from prestatario)
```

La operacion **except** (excepto) elimina duplicados automáticamente. Así, en la consulta anterior, una tupla con el nombre de Santos aparecerá en el resultado (exactamente una vez), sólo si Santos tiene una cuenta en el banco, pero no tiene ningún préstamo en el mismo.

Para conservar los duplicados, se utilizará **except all** en lugar de **except**:

```
(select nombre-cliente
 from impositor)
except all
(select nombre-cliente
 from prestatario)
```

El número de copias duplicadas de una tupla en el resultado es igual al número de copias duplicadas de dicha tupla en *i* menos el número de copias duplicadas de la misma tupla en *p*, siempre que la diferencia sea positiva. Así, si Santos tuviese tres cuentas y un préstamo en el banco, entonces en el resultado aparecerían dos tuplas con el nombre de Santos. Si, por el contrario, dicho cliente tuviese dos cuentas y tres préstamos en el banco, no habrá ninguna tupla con el nombre de Santos en el resultado.

4.4. FUNCIONES DE AGREGACIÓN

Las funciones de agregación son funciones que toman una colección (un conjunto o multiconjunto) de valores como entrada y producen un único valor como salida. SQL proporciona cinco funciones de agregación primitivas:

- Media: **avg**
- Mínimo: **min**
- Máximo: **max**
- Total: **sum**
- Cuenta: **count**

La entrada a **sum** y **avg** debe ser una colección de números, pero los otros operadores pueden operar sobre colecciones de datos de tipo no numérico, tales como las cadenas.

Como ejemplo, considérese la consulta «Obtener la media de saldos de las cuentas de la sucursal Navacerrada». Esta consulta se puede formular del modo siguiente:

select avg (*saldo*)
from *cuenta*
where *nombre-sucursal* = 'Navacerrada'

El resultado de esta consulta será una relación con un único atributo, que contendrá una única fila con un valor numérico correspondiente al saldo medio de la sucursal Navacerrada. Opcionalmente se puede dar un nombre al atributo resultado de la relación, usando la cláusula **as**.

Existen situaciones en las cuales sería deseable aplicar las funciones de agregación no sólo a un único conjunto de tuplas sino también a un grupo de conjuntos de tuplas; esto se especifica en SQL usando la cláusula **group by**. El atributo o atributos especificados en la cláusula **group by** se usan para formar grupos. Las tuplas con el mismo valor en todos los atributos especificados en la cláusula **group by** se colocan en un grupo.

Como ejemplo, considérese la consulta «Obtener el saldo medio de las cuentas de cada sucursal».

Dicha consulta se formulará del modo siguiente

select *nombre-sucursal*, **avg** (*saldo*)
from *cuenta*
group by *nombre-sucursal*

La conservación de duplicados es importante al calcular una media. Supóngase que los saldos de las cuentas en la (pequeña) sucursal de nombre «Galapagar» son 1.000 €, 3.000 €, 2.000 € y 1.000 €. El saldo medio es 7.000/4 =1.750 €. Si se eliminasen duplicados se obtendría un resultado erróneo (6.000/3 = 2.000 €).

Hay casos en los que se deben eliminar los duplicados antes de calcular una función de agregación. Para eliminar duplicados se utiliza la palabra clave **distinct** en la expresión de agregación. Como ejemplo considérese la consulta «Obtener el número de impositores de cada sucursal». En este caso un impositor sólo se debe contar una vez, sin tener en cuenta el número de cuentas que el impositor pueda tener. La consulta se formulará del modo siguiente:

select *nombre-sucursal*, **count** (**distinct** *nombre-cliente*)
from *impositor*, *cuenta*
where *impositor.número-cuenta* = *cuenta.número-cuenta*
group by *nombre-sucursal*

A veces es más útil establecer una condición que se aplique a los grupos que una que se aplique a las tuplas. Por ejemplo, podemos estar interesados sólo en aquellas sucursales donde el saldo medio de cuentas es superior a 1.200 €. Esta condición no es aplicable a una única tupla; se aplica a cada grupo construido por la cláusula **group by**. Para expresar este tipo de consultas se utiliza la cláusula **having** de SQL. Los predicados de la cláusula **having** se aplican después de la formación de grupos, de modo que se pueden usar las funciones de agregación. Esta consulta se expresa en SQL del modo siguiente:

select *nombre-sucursal*, **avg** (*saldo*)
from *cuenta*
group by *nombre-sucursal*
having avg (saldo) > 1200

A veces se desea tratar la relación entera como un único grupo. En casos de este tipo no se usa la cláusula **group by**. Considérese la consulta «Obtener el saldo medio de todas las cuentas». Esta consulta se formulará del modo siguiente:

select avg (*saldo*)
from *cuenta*

Con mucha frecuencia se usa la función de agregación **count** para contar el número de tuplas de una relación. La notación para esta función en SQL es **count** (*). Así, para encontrar el número de tuplas de la relación *cliente*, se escribirá

select count (*)
from *cliente*

SQL no permite el uso de **distinct** con **count** (*). Sí se permite, sin embargo, el uso de **distinct** con **max** y **min**, incluso cuando el resultado no cambia. Se puede usar la palabra clave **all** en lugar de **distinct** para especificar la retención de duplicados, pero como **all** se especifica de manera predeterminada, no es necesario incluir dicha cláusula.

Si en una misma consulta aparece una cláusula **where** y una cláusula **having**, se aplica primero el predicado de la cláusula **where**. Las tuplas que satisfagan el predicado de la cláusula **where** se colocan en grupos según la cláusula **group by**. La cláusula **having**, si existe, se aplica entonces a cada grupo; los grupos que no satisfagan el predicado de la cláusula **having** se eliminan. La cláusula **select** utiliza los grupos restantes para generar las tuplas resultado de la consulta.

Para ilustrar el uso de la cláusula **where** y la cláusula **having** dentro de la misma consulta considérese el ejemplo «Obtener el saldo medio de cada cliente que vive en Madrid y tiene como mínimo tres cuentas».

select *impositor.nombre-cliente*, **avg** (*saldo*)
from *impositor*, *cuenta*, *cliente*
where *impositor.número-cuenta*
= *cuenta.número-cuenta* **and**
impositor.nombre-cliente
= *cliente.nombre-cliente* **and**
ciudad-cliente = 'Madrid'
group by *impositor.nombre-cliente*
having count (**distinct** *impositor.número-cuenta*) >= 3

4.5. VALORES NULOS

SQL permite el uso de valores nulos para indicar la ausencia de información sobre el valor de un atributo.

En un predicado se puede usar la palabra clave especial **null** para comprobar si un valor es nulo. Así, para encontrar todos los números de préstamo que aparecen en la relación *préstamo* con valores nulos para *importe* se escribe

select *número-préstamo*
from *préstamo*
where *importe* **is null**

El predicado **is not null** pregunta por la ausencia de un valor nulo.

El uso de un valor nulo en las operaciones aritméticas y de comparación causa varias complicaciones. En el Apartado 3.3.4 se vio cómo se manejan los valores nulos en el álgebra relacional. Ahora se describe cómo maneja SQL los valores nulos.

El resultado de una expresión aritmética (incluyendo por ejemplo +,–,∗ o /) es nulo si cualquiera de los valores de entrada es nulo. SQL trata como **desconocido** el resultado de cualquier comparación que implique un valor *nulo* (aparte de **is null** e **is not null**).

Dado que el predicado en una cláusula **where** puede incluir operaciones booleanas tales como **and**, **or** y **not** sobre los resultados de las comparaciones, las definiciones de estas operaciones se extienden para manejar el valor **desconocido**, como se describe en el Apartado 3.3.4.

- **and**: el resultado de *cierto* **and** *desconocido* es *desconocido*, *falso* **and** *desconocido* es *falso*, mientras que *desconocido* **and** *desconocido* es *desconocido*.
- **or**: el resultado de *cierto* **or** *desconocido* es *cierto*, *falso* **or** *desconocido* es *desconocido*, mientras que *desconocido* **or** *desconocido* es *desconocido*.

SQL define el resultado de una instrucción SQL de la forma

select … from R_1, …., R_n **where** P

para contener (proyecciones de) tuplas en $R_1 \times \ldots \times R_n$ para las que el predicado P se evalúa a **cierto**. Si el predicado se evalúa a **falso** o **desconocido** para una tupla de $R_1 \times \ldots \times R_n$ (la proyección de) la tupla no se añade al resultado.

SQL también permite determinar si el resultado de una comparación es desconocido en lugar de cierto o falso usando las cláusulas **is unknown** (es desconocido) e **is not unknown** (no es desconocido)

La existencia de valores nulos también complica el procesamiento de los operadores de agregación. Supóngase que algunas tuplas en la relación *préstamo* tienen valor nulo para el atributo *importe*. Considérese en ese caso la siguiente consulta, que calcula el total de todas las cantidades prestadas:

select sum (*importe*)
from *préstamo*

Los valores que van a ser sumados en la consulta anterior incluyen valores nulos, puesto que algunas tuplas tienen valor nulo para el atributo *importe*. En lugar de decir que la suma total es nula, la norma SQL establece que el operador **sum** debería ignorar los valores nulos de su entrada.

En general, las funciones de agregación tratan los valores nulos según la regla siguiente: todas las funciones de agregación excepto **count(*)** ignoran los valores nulos de la colección de datos de entrada. Como resultado de ignorar los valores nulos, la colección de valores de entrada puede resultar vacía. El cálculo de **count** de una colección vacía se define como 0 y todas las demás operaciones de agregación devuelven un valor nulo cuando se aplican sobre una colección de datos vacía. El efecto de los valores nulos en algunas de las construcciones más complicadas de SQL puede ser más sutil.

En SQL:1999 se introdujo un tipo de datos **boolean**, que puede tomar los valores **cierto**, **falso** y **desconocido**. Las funciones de agregación **some** (algún) y **every** (cada), que significan exactamente lo que se espera de ellas, se pueden aplicar a una colección de valores booleanos.

4.6. SUBCONSULTAS ANIDADAS

SQL proporciona un mecanismo para las subconsultas anidadas. Una subconsulta es una expresión **select-from-where** que se anida dentro de otra consulta. Un uso común de subconsultas es llevar a cabo comprobaciones sobre pertenencia a conjuntos, comparación de conjuntos y cardinalidad de conjuntos. Estos usos se estudiarán en los apartados siguientes.

4.6.1. Pertenencia a conjuntos

SQL utiliza el cálculo relacional para las operaciones que permiten comprobar la pertenencia de una tupla a una relación. La conectiva **in** comprueba la pertenencia a un conjunto, donde el conjunto es la colección de valores resultado de una cláusula **select**. La conectiva

not in comprueba la no pertenencia a un conjunto. Como ejemplo considérese de nuevo la consulta «Encontrar todos los clientes que tienen tanto un préstamo como una cuenta en el banco». Anteriormente escribimos esta consulta como la intersección de dos conjuntos: el conjunto de los impositores del banco y el conjunto de los prestatarios del banco. Sin embargo, existe un enfoque alternativo consistente en encontrar todos los tenedores de cuentas en el banco que son miembros del conjunto de prestatarios. Claramente, esta formulación genera el mismo resultado que la anterior, pero obliga a formular la consulta usando la conectiva **in** de SQL. A continuación, se van a obtener todos los tenedores de cuentas formulando así la siguiente subconsulta:

(**select** *nombre-cliente*
 from *impositor*)

A continuación es necesario encontrar aquellos clientes que son prestatarios del banco y que aparecen en la lista de tenedores de cuenta, obtenida como resultado de la subconsulta anterior. Esto se consigue anidando la subconsulta en un **select** más externo. La consulta resultante es la siguiente:

select distinct *nombre-cliente*
from *prestatario*
where *nombre-cliente* **in** (**select** *nombre-cliente*
from *impositor*)

Este ejemplo muestra que es posible escribir la misma consulta de diversas formas en SQL. Esta flexibilidad es de gran utilidad, puesto que permite al usuario pensar en una consulta del modo que le parezca más natural. Más adelante se verá que existe una gran cantidad de redundancia en SQL.

En el ejemplo anterior se comprobaba la pertenencia a un conjunto en una relación de un solo atributo. También es posible comprobar la pertenencia a un conjunto en una relación cualquiera. Así, se puede formular la consulta «Listar los clientes que tienen tanto una cuenta como un préstamo en la sucursal Navacerrada» de un modo distinto al visto anteriormente:

select distinct *nombre-cliente*
from *prestatario*, *préstamo*
where *prestatario.número-préstamo* =
préstamo.número-préstamo **and**
nombre-sucursal = 'Navacerrada' **and**
(*nombre-sucursal*, *nombre-cliente*) **in**
(**select** *nombre-sucursal*, *nombre-cliente*
from *impositor*, *cuenta*
where *impositor.número-cuenta*
= *cuenta*. *número-cuenta*)

A continuación, se ilustra el uso de la constructora **not in**. Por ejemplo, para encontrar todos los clientes que tienen un préstamo en el banco, pero no tienen una cuenta en el banco, se puede escribir

select distinct *nombre-cliente*
from *prestatario*
where *nombre-cliente* **not in** (**select** *nombre-cliente*
from *impositor*)

Los operadores **in** y **not in** también se pueden usar sobre conjuntos enumerados. La consulta siguiente selecciona los nombres de los clientes que tienen un préstamo en el banco y cuyos nombres no son ni «Santos» ni «Gómez».

select distinct *nombre-cliente*
from *prestatario*
where *nombre-cliente* **not in** ('Santos', 'Gómez')

4.6.2. Comparación de conjuntos

Considérese la consulta «Obtener los nombres de todas las sucursales que poseen un activo mayor que al menos una sucursal situada en Barcelona». En el Apartado 4.2.5 se formulaba esta consulta del modo siguiente:

select distinct *T.nombre-sucursal*
from *sucursal* **as** *T*, *sucursal* **as** S
where *T.activo* > *S.activo* **and**
S.ciudad-sucursal = 'Barcelona'

SQL ofrece, sin embargo, un estilo alternativo de formular la consulta anterior. La expresión: «mayor que al menos una» se representa en SQL por > **some**. Esta constructora permite reescribir la consulta en una forma más parecida a la formulación de la consulta en lenguaje natural.

select *nombre-sucursal*
from *sucursal*
where *activo* > **some** (**select** *activo*
from *sucursal*
where *ciudad-sucursal*
= 'Barcelona')

La subconsulta

(**select** *activo*
from *sucursal*
where *ciudad-sucursal* = 'Barcelona')

genera el conjunto de todos los valores de activo para todas las sucursales sitas en Barcelona. La comparación > **some,** en la cláusula **where** de la cláusula **select** más externa, es cierta si el valor del atributo *activo* de la tupla es mayor que al menos un miembro del conjunto de todos los valores de *activo* de las sucursales de Barcelona.

SQL también permite realizar las comparaciones < **some**, <= **some**, >= **some**, = **some** y <> **some**. Como ejer-

cicio, se puede verificar que **= some** es idéntico a **in**, mientras que <= **some** no es lo mismo que **not in**. En SQL, la palabra clave **any** es sinónimo de **some**. Las versiones más antiguas de SQL sólo admitían **any**. Sin embargo, versiones posteriores añadieron la alternativa **some** para evitar la ambigüedad lingüística de la palabra inglesa *any*.

Ahora la consulta se modificará ligeramente a fin de obtener los nombres de todas las sucursales que tienen un activo superior al de todas las sucursales de Barcelona. La constructora > **all** corresponde a la expresión «superior a todas». Utilizando esta constructora la consulta se podría formular del modo siguiente:

```
select nombre-sucursal
from sucursal
where activo > all (select activo
                    from sucursal
                    where ciudad-sucursal
                          = 'Barcelona')
```

Al igual que con **some**, SQL también permite utilizar las comparaciones < **all**, <= **all**, >= **all**, = **all** y <> **all**. Como ejercicio se puede verificar que <= **any** es lo mismo que **not in.**

Como otro ejemplo de comparaciones considérese la consulta «Encontrar la sucursal que tiene el mayor saldo medio». En SQL, las funciones de agregación no se pueden componer. Así, no está permitido el uso de **max** (**avg** (…)). Por ello, para la formulación de esta consulta se seguirá la estrategia siguiente: para empezar se formula una consulta para encontrar todos los saldos medios, y luego se anida ésta como subconsulta de una consulta más larga que encuentre aquellas sucursales para las que el saldo medio es mayor o igual que todos los saldos medios:

```
select nombre-sucursal
from cuenta
group by nombre-sucursal
having avg (saldo) >= all (select avg (saldo)
                           from cuenta
                           group by nombre-sucursal)
```

4.6.3. Comprobación de relaciones vacías

SQL incluye la posibilidad de comprobar si una subconsulta no produce ninguna tupla como resultado. La constructora **exists** devuelve el valor **cierto** si la subconsulta argumento no es vacía.

Usando la constructora **exists** se puede formular la consulta «Obtener los clientes que tienen tanto una cuenta como un préstamo en el banco» de otra nueva forma:

```
select nombre-cliente
where exists (select *
              from impositor
              where impositor.nombre-cliente =
                    prestatario.nombre-cliente)
```

Utilizando la constructora **not exists** se puede comprobar la inexistencia de tuplas en el resultado de una subconsulta. Además, es posible usar la constructora **not exists** para simular la operación de continencia de conjuntos (es decir, superconjunto). Así, se puede escribir la expresión «la relación *A* contiene a la relación *B*» como «**not exists** (B **except** A)». Aunque no forma parte de SQL estándar, el operador **contains** aparece en algunos sistemas relacionales. Para ilustrar el operador **not exists** considérese otra vez la consulta «Obtener todos los clientes que tienen una cuenta en todas las sucursales de Barcelona». Será necesario comprobar para cada cliente si el conjunto de todas las sucursales en las que dicho cliente tiene cuenta contiene al conjunto de todas las sucursales de Barcelona. Utilizando el operador **except** se puede formular la consulta del modo siguiente:

```
select distinct S.nombre-cliente
from impositor as S
where not exists ((select nombre-sucursal
                   from sucursal
                   where ciudad-sucursal
                         = 'Barcelona')
                  except
                   (select R.nombre-sucursal
                    from impositor as T, cuenta as R
                    where T.número-cuenta
                          = R.número-cuenta and
                          S.nombre-cliente
                          = T.nombre-cliente ))
```

En este ejemplo, la subconsulta

```
(select nombre-sucursal
 from sucursal
 where ciudad-sucursal = 'Barcelona')
```

obtiene todas las sucursales de Barcelona. Por otro lado, la subconsulta

```
(select R.nombre-sucursal
 from impositor as T, cuenta as R
 where T.número-cuenta = R.número-cuenta and
       S.nombre-cliente = T.nombre-cliente )
```

obtiene todas las sucursales en las cuales el cliente *S.nombre-cliente* tiene una cuenta. Por último, el **select** más externo toma cada cliente y comprueba si el conjunto de todas las sucursales en las que dicho cliente tiene cuenta, contiene al conjunto de todas las sucursales de Barcelona.

En consultas que contengan subconsultas se aplica una regla de visibilidad para las variables tupla. En una subconsulta, sólo se pueden usar variables tupla que estén definidas en la propia subconsulta o en cualquier consulta que contenga a dicha subconsulta. Si una variable tupla está definida tanto localmente (en una sub-

consulta) como globalmente (en una consulta que contenga a la subconsulta) se aplica la definición local. Esta regla es análoga a la utilizada para las variables en los lenguajes de programación.

4.6.4. Comprobación de tuplas duplicadas

SQL incluye la posibilidad de comprobar si una subconsulta produce como resultado tuplas duplicadas. La constructora **unique** devuelve el valor **cierto** si la subconsulta que se le pasa como argumento no produce tuplas duplicadas. Usando la constructora **unique** se puede formular la consulta «Obtener todos los clientes que tienen sólo una cuenta en la sucursal de nombre Navacerrada» del siguiente modo:

```
select T.nombre-cliente
from impositor as T
where unique (select R.nombre-cliente
              from cuenta, impositor as R
              where T.nombre-cliente
                    = R.nombre-cliente and
                    R.número-cuenta
                    = cuenta.número-cuenta and
                    cuenta.nombre-sucursal
                    = 'Navacerrada')
```

La existencia de tuplas duplicadas en una subconsulta se puede comprobar utilizando la constructora **not unique**. Para ilustrar esta constructora considérese la consulta «Obtener todos los clientes que tienen al menos dos cuentas en la sucursal Navacerrada», que se puede formular del modo siguiente:

```
select distinct T.nombre-cliente
from impositor as T
where not unique (select R.nombre-cliente
                  from cuenta, impositor as R
                  where T.nombre-cliente
                        = R.nombre-cliente and
                        R.número-cuenta =
                        cuenta.número-cuenta and
                        cuenta.número-sucursal
                        = 'Navacerrada')
```

Formalmente, la comprobación hecha por la constructora **unique** sobre una relación debería fallar si y sólo si en la relación existieran dos tuplas t_1 y t_2 tales que $t_1 = t_2$. Como la comprobación $t_1 = t_2$ sólo falla si cualquier campo de t_1 o de t_2 es nulo, entonces es posible que el resultado de **unique** sea **cierto** incluso si existen varias copias de una tupla, siempre que al menos uno de los atributos de la tupla sea nulo.

4.7. VISTAS

Una vista en SQL se define utilizando la orden **create view**. Para definir una vista se le debe dar un nombre y se debe construir la consulta que genere dicha vista. La forma de la orden **create view** es la siguiente:

create view *v* **as** <expresión de consulta>

donde <expresión de consulta> puede ser cualquier consulta válida. El nombre de la vista se representa por *v*. Nótese que la notación usada para la definición de una vista en el álgebra relacional (véase Capítulo 3) se basa en esta de SQL.

Como ejemplo considérese la vista consistente en los nombres de sucursales y los nombres de los clientes que tienen una cuenta o un préstamo en esa sucursal. Si se denomina esta vista como *todos-los-clientes* se definirá del modo siguiente:

```
create view todos-los-clientes as
    (select nombre-sucursal, nombre-cliente
     from impositor, cuenta
     where impositor.número-cuenta
           = cuenta.número-cuenta)
  union
    (select nombre-sucursal, nombre-cliente
     from prestatario, préstamo
     where prestatario.número-préstamo
           = préstamo.número-préstamo)
```

Los nombres de los atributos de una vista se pueden indicar explícitamente de la forma siguiente:

```
create view total-préstamos-sucursal
            (nombre-sucursal, total-préstamos) as
select nombre-sucursal, sum (importe)
from préstamo
group by nombre-sucursal
```

La vista anterior contiene para cada sucursal la suma de los importes de todos los préstamos de esa sucursal. Como la expresión **sum** (*importe*) no tiene nombre, el nombre del atributo se especifica explícitamente en la definición de la vista.

Los nombres de vistas pueden aparecer en cualquier lugar en el que pudiera aparecer un nombre de relación. Usando la vista *todos-los-clientes*, se pueden listar todos los clientes de la sucursal Navacerrada, escribiendo

```
select nombre-cliente
from todos-los-clientes
where nombre-sucursal = 'Navacerrada'
```

4.8. CONSULTAS COMPLEJAS

Las consultas complejas son a menudo difíciles o imposibles de escribir como un único bloque SQL o una unión, intersección o diferencia de bloques SQL (un bloque SQL consiste en una única instrucción **select from where**, posiblemente con cláusulas **group by** y **having**). Aquí se estudian dos formas de componer varios bloques SQL para expresar una consulta compleja: las relaciones derivadas y la cláusula **with**.

4.8.1. Relaciones derivadas

SQL permite el uso de una expresión de subconsulta en la cláusula **from**. Si se usa una expresión de este tipo se debe dar un nombre a la relación resultado y se pueden renombrar los atributos usando la cláusula **as**. Por ejemplo, considérese la subconsulta

```
(select nombre-sucursal, avg (saldo)
 from cuenta
 group by nombre-sucursal)
as media-sucursal (nombre-sucursal, saldo-medio)
```

Esta subconsulta produce una relación consistente en los nombres de todas las sucursales y sus correspondientes saldos de cuenta medios. El resultado de la subconsulta recibe el nombre de *media-sucursal* y contiene los atributos *nombre-sucursal* y *saldo-medio*.

Para ilustrar el uso de una expresión de subconsulta en la cláusula **from** considérese la consulta «Obtener el saldo medio de las cuentas de aquellas sucursales donde dicho saldo medio sea superior a 1.200 €». En el Apartado 4.4 se formulaba esta consulta utilizando la cláusula **having**. Ahora se puede reescribir dicha consulta sin usar esta cláusula de la siguiente forma:

```
select nombre-sucursal, saldo-medio
from (select nombre-sucursal, avg (saldo)
        from cuenta
        group by nombre-sucursal)
      as resultado (nombre-sucursal, saldo-medio)
where saldo-medio > 1200
```

En esta formulación no es necesario el uso de la cláusula **having** puesto que la relación temporal *resultado* se calcula en la cláusula **from**, y los atributos de *resultado* se pueden usar directamente en la cláusula **where**.

Supóngase como otro ejemplo que se desea hallar el máximo del total de saldos de todas las sucursales. La cláusula **having** no sirve en este caso, pero se puede escribir fácilmente esta consulta usando una subconsulta en la cláusula **from**, como se muestra a continuación:

```
select max(saldo-total)
from (select nombre-sucursal, sum(saldo)
      from cuenta
      group by nombre-sucursal) as
            total-sucursal(nombre-sucursal,
            saldo-total)
```

4.8.2. La cláusula with

Las consultas complicadas son mucho más fáciles de formular y de entender si se descomponen en vistas más simples y después se combinan, al igual que se estructuran los programas, descomponiendo sus tareas en procedimientos. Sin embargo, son distintas a la definición de procedimientos en cuanto a que una cláusula **create view** crea una definición de vista en la base de datos y esa definición de vista permanece en la base de datos hasta que se ejecuta una orden **drop view** *nombre-vista*.

La cláusula **with** proporciona una forma de definir una vista temporal cuya definición está disponible sólo para la consulta en la que aparece esta cláusula. Considérese la siguiente consulta, que selecciona cuentas con el saldo máximo; si hay muchas cuentas con el mismo saldo máximo, todas ellas se seleccionan.

```
with saldo-máximo(valor) as
     select max (saldo)
     from cuenta
select número-cuenta
from cuenta, saldo-máximo
where cuenta.saldo = saldo-máximo.valor
```

La cláusula **with** introducida en SQL:1999 se incluye actualmente sólo en algunas bases de datos.

Se podría haber escrito la consulta anterior usando una subconsulta anidada tanto en la cláusula **from** como en la **where**. Sin embargo, el uso de subconsultas anidadas hace que la consulta sea más difícil de leer y entender. La cláusula **with** hace que la lógica de la consulta sea más clara; también permite usar una definición de vista en varios lugares de una consulta.

Por ejemplo, supóngase que se desea encontrar todas las sucursales donde el depósito de cuentas es mayor que la media del total de depósitos de cuentas en todas las sucursales. Se puede escribir la consulta con la cláusula **with** como se muestra a continuación.

```
with total-sucursal(nombre-sucursal,valor) as
     select nombre-sucursal, sum(saldo)
     from cuenta
     group by nombre-sucursal
with total-media-sucursal(valor) as
     select avg(sa valor ldo)
     from total-sucursal
```

select *nombre-sucursal*
from *total-sucursal, total-media-sucursal*
where *total-sucursal.valor* >= *total-media-sucursal.valor*

Por supuesto, se puede crear una consulta equivalente sin la cláusula **with**, pero sería más complicada y difícil de entender. Como ejercicio, se puede escribir la consulta equivalente.

4.9. MODIFICACIÓN DE LA BASE DE DATOS

Hasta ahora nos hemos limitado a la extracción de información de una base de datos. A continuación se mostrará cómo añadir, eliminar o cambiar información utilizando SQL.

4.9.1. Borrado

Un borrado se expresa de igual modo que una consulta. Se pueden borrar sólo tuplas completas, es decir, no se pueden borrar valores de atributos concretos. Un borrado se expresa en SQL del modo siguiente:

delete from *r*
where *P*

donde *P* representa un predicado y *r* representa una relación. La declaración **delete** selecciona primero todas las tuplas *t* en *r* para las que *P* (*t*) es cierto y a continuación las borra de *r*. La cláusula **where** se puede omitir, en cuyo caso se borran todas las tuplas de *r*.

Hay que señalar que una orden **delete** opera sólo sobre una relación. Si se desea borrar tuplas de varias relaciones, se deberá utilizar una orden **delete** por cada relación. El predicado de la cláusula **where** puede ser tan complicado como el **where** de cualquier cláusula **select**, o tan simple como una cláusula **where** vacía. La consulta

delete from *préstamo*

borra todas las tuplas de la relación *préstamo* (los sistemas bien diseñados requerirán una confirmación del usuario antes de ejecutar una consulta tan devastadora).

A continuación se muestran una serie de ejemplos de borrados en SQL.

- Borrar todas las cuentas de la sucursal Navacerrada.

  delete from *cuenta*
  where *nombre-sucursal* = 'Navacerrada'

- Borrar todos los préstamos en los que la cantidad esté comprendida entre 1.300 € y 1.500 €.

  delete from *préstamo*
  where *importe* **between** 1300 **and** 1500

- Borrar las cuentas de todas las sucursales de Navacerrada.

  delete from *cuenta*
  where *nombre-sucursal* **in** (**select** *nombre-sucursal*
  from *sucursal*
  where *ciudad-sucursal*
  = 'Navacerrada')

  El borrado anterior selecciona primero todas las sucursales con sede en Navacerrada y a continuación borra todas las tuplas *cuenta* pertenecientes a esas sucursales.

Nótese que, si bien sólo se pueden borrar tuplas de una sola relación cada vez, se puede utilizar cualquier número de relaciones en una expresión **select-from-where** anidada en la cláusula **where** de un **delete**. La orden **delete** puede contener un **select** anidado que use una relación de la cual se van a borrar tuplas. Por ejemplo, para borrar todas las cuentas cuyos saldos sean inferiores a la media del banco se puede escribir:

delete from *cuenta*
where *saldo* < (**select avg** (*saldo*)
from *cuenta*)

La orden **delete** comprueba primero cada tupla de la relación *cuenta* para comprobar si la cuenta tiene un saldo inferior a la media del banco. A continuación se borran todas las tuplas que no cumplan la condición anterior, es decir, las que representan una cuenta con un saldo menor que la media. Es importante realizar todas las comprobaciones antes de llevar a cabo ningún borrado (si se borrasen algunas tuplas antes de que otras fueran comprobadas, el saldo medio podría haber cambiado y el resultado final del borrado dependería del orden en que las tuplas fueran procesadas).

4.9.2. Inserción

Para insertar datos en una relación, o bien se especifica la tupla que se desea insertar o se formula una consulta cuyo resultado sea el conjunto de tuplas que se desean insertar. Obviamente, los valores de los atributos de la tuplas que se inserten deben pertenecer al dominio de los atributos. De igual modo, las tuplas insertadas deberán ser de la aridad correcta.

La instrucción **insert** más sencilla corresponde a la de inserción de una tupla. Supongamos que se desea insertar en la base de datos el hecho de que hay una

cuenta C-9732 en la sucursal Navacerrada y que dicha cuenta tiene un saldo de 1.200 €. La inserción se puede formular del modo siguiente:

```
insert into cuenta
    values ('C-9732', 'Navacerrada', 1200)
```

En este ejemplo los valores se especifican en el mismo orden en que los atributos se listan en el esquema de relación. Para beneficio de los usuarios, que pueden no recordar el orden de los atributos, SQL permite que los atributos se especifiquen en la cláusula **insert**. Así, el siguiente ejemplo tiene una función idéntica al anterior:

```
insert into cuenta (nombre-sucursal, número-
        cuenta, saldo)
    values ('Navacerrada', 'C-9732', 1200)

insert into cuenta (número-cuenta, nombre-
        sucursal, saldo)
    values ('C-9732', 'Navacerrada', 1200)
```

Generalmente se desea insertar las tuplas que resultan de una consulta. Por ejemplo, si a todos los clientes tenedores de préstamos en la sucursal Navacerrada se les quisiera regalar, como gratificación, una cuenta de ahorro con 200 € por cada cuenta de préstamo que tienen, se podría escribir:

```
insert into cuenta
    select nombre-sucursal, número-préstamo, 200
    from préstamo
    where nombre-sucursal = 'Navacerrada'
```

En lugar de especificar una tupla, como se hizo en los primeros ejemplos de este apartado, se utiliza una instrucción **select** para especificar un conjunto de tuplas. La instrucción **select** se evalúa primero, produciendo un conjunto de tuplas que a continuación se insertan en la relación *cuenta*. Cada tupla tiene un *nombre-sucursal* (Navacerrada), un *número-préstamo* (que sirve como número de cuenta para la nueva cuenta) y un saldo inicial de la cuenta (200 €).

Además es necesario añadir tuplas a la relación *impositor*; para hacer esto, se escribirá:

```
insert into impositor
    select nombre-cliente, número-préstamo
    from prestatario, préstamo
    where prestatario.número-préstamo
        = préstamo.número-préstamo and
        nombre-sucursal = 'Navacerrada'
```

Esta consulta inserta en la relación *impositor* una tupla (*nombre-cliente*, *número-préstamo*) por cada *nombre-cliente* que posea un préstamo en la sucursal Navacerrada, con número de préstamo *número-préstamo*.

Es importante que la evaluación de la instrucción **select** finalice completamente antes de llevar a cabo ninguna inserción. Si se realizase alguna inserción antes de que finalizase la evaluación de la instrucción **select**, una consulta del tipo:

```
insert into cuenta
    select *
    from cuenta
```

podría insertar un número infinito de tuplas. La primera tupla de la relación *cuenta* se insertaría de nuevo en *cuenta*, creando así una segunda copia de la tupla. Como esta segunda copia ya sería parte de *cuenta*, la instrucción **select** podría seleccionarla, insertando así una tercera copia en la relación *cuenta*. Esta tercera copia podría ser seleccionada a continuación por el **select** e insertar una cuarta copia y así infinitamente. Evaluando completamente toda la instrucción **select** antes de realizar ninguna inserción se evitan este tipo de problemas.

Por ahora, en el estudio de la instrucción **insert** sólo se han considerado ejemplos en los que se especificaba un valor para cada atributo de las tuplas insertadas. Como se estudió en el Capítulo 3, es posible indicar sólo valores para algunos de los atributos del esquema. A cada uno de los atributos restantes, se les asignará un valor nulo, que se denota por *null*. Como ejemplo considérese la consulta:

```
insert into cuenta
    values ('C-401', null, 1200)
```

en la que se sabe que la cuenta C-401 tiene un saldo de 1.200 €, pero no se conoce el nombre de la sucursal. Considérese ahora la consulta

```
select número-cuenta
from cuenta
where nombre-sucursal = 'Navacerrada'
```

Como el nombre de la sucursal de la cuenta C-401 es desconocido, no se puede determinar si es igual a «Navacerrada».

Se puede prohibir la inserción de valores nulos utilizando el LDD de SQL, que se estudia en el Apartado 4.11.

4.9.3. Actualizaciones

En determinadas situaciones puede ser deseable cambiar un valor dentro de una tupla, sin cambiar *todos* los valores de la misma. Para este tipo de situaciones se utiliza la instrucción **update**. Al igual que ocurre con **insert** y **delete**, se pueden elegir las tuplas que van a ser actualizadas mediante una consulta.

Por ejemplo, si hubiera que realizar el pago de intereses anuales y todos los saldos se incrementasen en un 5 %, habría que formular la siguiente actualización:

```
update cuenta
set saldo = saldo * 1.05
```

Esta actualización se aplica una vez a cada tupla de la relación *cuenta*.

Si se paga el interés sólo a las cuentas con un saldo de 1.000 € o superior, se puede escribir

```
update cuenta
set saldo = saldo * 1.05
where saldo >= 1000
```

En general, la cláusula **where** de la instrucción **update** puede contener cualquier constructor legar en la cláusula **where** de una instrucción **select** (incluyendo instrucciones **select** anidadas). Como con **insert** y **delete**, un **select** anidado en una instrucción **update** puede referenciar la relación que se esté actualizando. Como antes, SQL primero comprueba todas las tuplas de la relación para determinar las que se deberían actualizar y después realiza la actualización. Por ejemplo, se puede escribir «Pagar un interés del 5% a las cuentas cuyo saldo sea mayor que la media» como sigue:

```
update cuenta
set saldo = saldo * 1.05
where (saldo > select avg(saldo)
              from cuenta)
```

Si se supone que las cuentas con saldos superiores a 10.000 € reciben un 6% de interés, mientras que las demás un 5%, se deberán escribir dos instrucciones de actualización:

```
update cuenta
set saldo = saldo * 1.06
where saldo > 10000

update cuenta
set saldo = saldo * 1.05
where saldo <= 10000
```

Obsérvese que, como se vio en el Capítulo 3, el orden en el que se ejecutan dos instrucciones de actualización es importante. Si se invierte el orden de las dos instrucciones anteriores, una cuenta con un saldo igual o muy poco inferior a 10.000 € recibiría un 11,3% de interés.

SQL ofrece una constructora **case**, que se puede usar para formular las dos instrucciones de actualización anteriores en una única instrucción de actualización, evitando el problema del orden de actualización.

```
update cuenta
set saldo = case
              when saldo <= 10000 then saldo * 1.05
              else saldo * 1.06
            end
```

La forma general de la instrucción **case** es la siguiente:

```
case
    when pred1 then result1
    when pred2 then result2
    …
    when predn then resultn
    else result0
end
```

La operación devuelve $result_i$, donde *i* es el primero de $result_1, result_2, \ldots, result_n$ que se satisface; si ninguno de ellos se satisface, la operación devuelve $result_0$. Las instrucciones **case** se pueden usar en cualquier lugar donde se espere un valor.

4.9.4. Actualización de vistas

La anomalía de la actualización de vistas estudiada en el Capítulo 3 también se produce en SQL. Como ejemplo considérese la siguiente definición de vista:

```
create view préstamo-sucursal as
    select nombre-sucursal, número-préstamo
    from préstamo
```

Como SQL permite que el nombre de una vista aparezca en cualquier lugar en el que pueda aparecer el nombre de una relación, se puede formular:

```
insert into préstamo-sucursal
    values ('Navacerrada', 'P-307')
```

SQL representa esta inserción mediante una inserción en la relación *préstamo*, puesto que *préstamo* es la relación real a partir de la cual se construye la vista *préstamo-sucursal*. Por lo tanto, debería especificarse un valor para el atributo *importe*. Este valor es un valor nulo. De este modo, la inserción anterior es equivalente a la inserción de la tupla

('P-307', 'Navacerrada', *null*)

en la relación *préstamo*.

Como vimos en el Capítulo 3, la anomalía de la actualización de vistas se agrava cuando una vista se define en términos de varias relaciones. Como resultado, muchas bases de datos basadas en SQL imponen la siguiente restricción a las modificaciones de vistas:

- Una modificación de una vista es válida sólo si la vista en cuestión se define en términos de la base de datos relacional real, esto es, del nivel lógico de la base de datos (y sin usar agregación).

Bajo esta restricción, las operaciones **update**, **insert** y **delete** realizadas sobre el ejemplo de la vista *todos-los-clientes* definida anteriormente, estarían prohibidas.

4.9.5. Transacciones

Una **transacción** consiste en una secuencia de instrucciones de consulta y actualizaciones. La norma SQL especifica que una transacción comienza implícitamente cuando se ejecuta una instrucción SQL. Una de las siguientes instrucciones SQL debe finalizar la transacción:

- **Commit work** compromete la transacción actual; es decir, hace que los cambios realizados por la transacción sean permanentes en la base de datos. Después de que se comprometa la transacción se inicia una nueva transacción automáticamente.
- **Rollback work** causa el retroceso de la transacción actual; es decir, deshace todas las actualizaciones realizadas por las instrucciones SQL de la transacción; así, el estado de la base de datos se restaura al que existía previo a la ejecución de la transacción.

La palabra **work** es opcional en ambas instrucciones.

El retroceso de transacciones es útil si se detecta alguna condición de error durante la ejecución de una transacción. El compromiso es similar, bajo un punto de vista, a guardar los cambios de un documento que se esté modificando, mientras que el retroceso es similar a abandonar la sesión de modificación sin guardar los cambios. Una vez que una transacción haya ejecutado **commit work**, sus efectos no se pueden deshacer con **rollback work**. El sistema de bases de datos garantiza que en el caso de una caída, los efectos de la transacción se retrocederán si no se hubo ejecutado **commit work**. En el caso de fallo de alimentación o caída del sistema, el retroceso ocurre cuando el sistema se reinicia.

Por ejemplo, para transferir dinero de una cuenta a otra es necesario actualizar dos saldos de cuenta. Las dos instrucciones de actualización formarían una transacción. Un error durante la ejecución de una de las instrucciones de una transacción resultaría en deshacer los efectos de las instrucciones anteriores de la transacción, de manera que la base de datos no se quedase en un estado parcialmente actualizado. En el Capítulo 15 se estudian más propiedades de las transacciones.

Si un programa termina sin ejecutar ninguna de estas órdenes, las actualizaciones se comprometen o retroceden. La norma no especifica cuál de los dos se debe realizar, con lo que la elección es dependiente de la implementación. En muchas implementaciones SQL, cada instrucción SQL es de manera predeterminada una transacción y se compromete tan pronto como se ejecuta. El compromiso automático de las instrucciones SQL individuales se puede desactivar si es necesario ejecutar una transacción que conste de varias instrucciones SQL. La forma de desactivación del compromiso automático depende de la implementación SQL específica.

Una alternativa mejor, que es parte de la norma SQL:1999 (pero actualmente sólo soportada por algunas implementaciones SQL), es permitir encerrar varias instrucciones SQL entre las palabras clave **begin atomic … end**. Todas las instrucciones entre las palabras clave forman así una única transacción.

4.10. REUNIÓN DE RELACIONES**

Además de proporcionar el mecanismo básico de producto cartesiano para reunir tuplas de relaciones, permitido en versiones anteriores, SQL también proporciona varios mecanismos para reunir relaciones, incluyendo reuniones condicionales y reuniones naturales, así como varias formas de reunión externa. Estas operaciones adicionales se usan a menudo como subconsultas dentro de la cláusula **from**.

4.10.1. Ejemplos

En la Figura 4.1 se muestran las relaciones *préstamo* y *prestatario* que usarán las distintas operaciones de reunión que ilustraremos. Comenzaremos con un ejemplo simple de reunión interna. En la Figura 4.2 se muestra el resultado de la expresión:

préstamo **inner join** *prestatario* **on**
préstamo.número-préstamo
= prestatario.número-préstamo

La expresión calcula la reunión zeta de las relaciones *préstamo* y *prestatario*, donde la condición de reunión es *préstamo.número-préstamo = prestatario.número-préstamo*. Los atributos del resultado son los atributos del lado izquierdo de la relación, seguidos por los del lado derecho de la misma.

número-préstamo	*nombre-sucursal*	*importe*
P-170	Centro	3.000
P-230	Moralzarzal	4.000
P-260	Navacerrada	1.700

préstamo

nombre-cliente	*número-préstamo*
Santos	P-170
Gómez	P-230
López	P-155

prestatario

FIGURA 4.1. Las relaciones *préstamo* y *prestatario*.

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*	*número-préstamo*
P-170	Centro	3.000	Santos	P-170
P-230	Moralzarzal	4.000	Gómez	P-230

FIGURA 4.2. Resultado de *préstamo* **inner join** *prestatario* **on** *préstamo.número-préstamo = prestatario.número-préstamo*.

Obsérvese que el atributo *número-préstamo* aparece dos veces en la figura (la primera aparición es debida a la relación *préstamo* y la segunda a *prestatario*). La norma SQL no requiere que los nombres de atributo en los resultados sean únicos. Se debería usar una cláusula **as** para asignar nombres únicos de atributos en los resultados de las consultas y subconsultas.

Renombramos la relación resultado de una reunión y los atributos de la relación resultado utilizando la cláusula **as**, como se ilustra a continuación:

préstamo **inner join** *prestatario* **on**
préstamo.número-préstamo
= prestatario.número-préstamo
as *ps* (*sucursal*, *número-préstamo*, *importe*,
cliente, *número-préstamo-cliente*)

La segunda aparición de *número-préstamo* se ha renombrado como *número-préstamo-cliente*. El orden de los atributos en el resultado de una reunión natural es importante a la hora de renombrarlos.

A continuación se muestra un ejemplo del uso de la operación reunión externa por la izquierda (**left outer join**):

préstamo **left outer join** *prestatario* **on**
préstamo.número-préstamo
= prestatario.número-préstamo

La reunión externa por la izquierda se calcula del modo siguiente. Primero se calcula el resultado de la reunión interna, como se vio anteriormente. A continuación, para cada tupla *t* de la unión interna, perteneciente a la relación del lado izquierdo (*préstamo*) que no encaje con ninguna tupla de la relación del lado derecho (*prestatario)*, se añade al resultado de la reunión una tupla *r*, como se indica a continuación. Los atributos de la tupla *r* que se derivan de la relación del lado izquierdo se rellenan con los valores de la tupla *t* y el resto de los atributos de *r* se rellenan con valores nulos. La relación resultante se muestra en la Figura 4.3. Las tuplas (P-170, Centro, 3.000) y (P-230, Moralzarzal, 4.000) se reúnen con las tuplas de *prestatario* y aparecen en el resultado de la reunión interna, y por ello en el resultado de la reunión externa por la izquierda. Por otra parte, la tupla (P-260, Navacerrada, 1.700) no encaja con ninguna tupla de *prestatario* en la reunión natural y por eso en el resultado de la reunión externa por la izquierda aparece la tupla (P-260, Navacerrada, 1.700, null, null).

Finalmente, se incluye un ejemplo del uso de la operación reunión natural (**natural join**).

préstamo **natural inner join** *prestatario*

Esta expresión calcula la reunión natural de dos relaciones. El único nombre de atributo común en *préstamo* y *prestatario* es *número-préstamo*. El resultado de la expresión anterior se muestra en la Figura 4.4. Este resultado es similar al resultado de la reunión interna con la condición **on** mostrada en la Figura 4.2, puesto que tienen la misma condición de reunión. Sin embargo, el atributo *número-préstamo* aparece sólo una vez en el resultado de la reunión natural, mientras que aparece dos veces en el resultado de la reunión con la condición **on**.

4.10.2 Tipos y condiciones de reunión

En el Apartado 4.10.1 se muestran ejemplos de los operadores de reunión incluidos en SQL-92. Las operaciones de reunión toman como entrada dos relaciones y devuelven como resultado otra relación. Aunque las expresiones de reunión externa se usan normalmente en la cláusula **from**, se pueden utilizar en cualquier lugar en el que cabría usar cualquier otra relación.

Cada variante de las operaciones de reunión en SQL-92 está formado por un *tipo de reunión* y una *condición de reunión*. La condición de reunión indica las tuplas pertenecientes a las dos relaciones que encajan y los atributos que se incluyen en el resultado de la reunión. El tipo de reunión define cómo se tratan las tuplas de cada relación que no encajan con ninguna tupla de la otra relación (basado en la condición de reunión). En la Figu-

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*	*número-préstamo*
P-170	Centro	3.000	Santos	P-170
P-230	Moralzarzal	4.000	Gómez	P-230
P-260	Navacerrada	1.700	*null*	*null*

FIGURA 4.3. Resultado de *préstamo* **left outer join** *prestatario* **on** *préstamo.número-préstamo = prestatario.número-préstamo*.

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*
P-170	Centro	3.000	Santos
P-230	Moralzarzal	4.000	Gómez

FIGURA 4.4. Resultado de *préstamo* **inner join** *prestatario*.

ra 4.5 se muestran algunos de los tipos y condiciones de reunión. El primer tipo de reunión es la reunión interna y los otros tres son tres tipos de reuniones externas. Las tres condiciones de reunión son: la reunión **natural** y la condición **on**, ya vistas anteriormente, y la condición **using**, que se verá más adelante.

El uso de una condición de reunión es obligatorio en las reuniones externas, pero es opcional en las reuniones internas (ya que si se omite, el resultado será el producto cartesiano de las dos relaciones). La palabra clave **natural** aparece sintácticamente delante del tipo de reunión, como se mostró anteriormente, mientras que las condiciones **on** y **using** aparecen al final de la expresión de reunión. Las palabras clave **inner** y **outer** son opcionales, ya que el resto del tipo de reunión permite deducir si la reunión es una reunión interna o externa.

El significado de la condición de reunión **natural**, en términos de las tuplas de las relaciones que encajan, es inmediato. La ordenación de los atributos, dentro del resultado de la reunión natural es el siguiente: los atributos de reunión (es decir, los atributos comunes a las dos relaciones) aparecen en primer lugar, en el orden en el que aparezcan en la relación del lado izquierdo. A continuación están los demás atributos que no son de reunión de la relación del lado izquierdo y, al final, todos los atributos que no son de reunión de la relación del lado derecho de la relación.

El tipo de relación reunión externa por la derecha (**right outer join**) es simétrico al de reunión externa por la izquierda (**left outer join**). Las tuplas de la relación del lado derecho que no encajen con ninguna tupla de la relación del lado izquierdo se rellenan con valores nulos y se añaden al resultado de la reunión externa por la derecha.

La siguiente expresión es un ejemplo de la combinación de la condición de reunión natural con el tipo de reunión externa por la derecha.

préstamo **natural right outer join** *prestatario*

El resultado de la expresión anterior se muestra en la Figura 4.6. Los atributos del resultado se definen por el tipo de reunión, que es una reunión natural; así, *número-préstamo* aparece sólo una vez. Las primeras dos tuplas del resultado provienen de la reunión natural de *préstamo* y *prestatario*. La tupla de la relación del lado derecho (López, P-155) no encaja en la reunión interna con ninguna tupla de la relación del lado izquierdo (*préstamo*). Así, la tupla (P-155, null, López) aparece en el resultado de la reunión.

Tipos de reunión	Condiciones de reunión
inner join **left outer join** **right outer join** **full outer join**	**natural** **on** <predicado> **using** (A_1, A_2, ..., A_n)

FIGURA 4.5. Tipos y condiciones de reunión.

La condición de reunión **using** ($A_1, A_2, \ldots, A_n$) es similar a la condición de reunión natural, salvo en que los atributos de reunión en este caso son A_1, A_2,…,A_n, en lugar de todos los atributos comunes de ambas relaciones y aparecen sólo una vez en el resultado de la unión.

El tipo de reunión externa completa (**full outer join**) es una combinación de los tipos de reunión externa por la derecha y por la izquierda. Después de calcular el resultado de la reunión interna, las tuplas de la relación del lado izquierdo que no encajen con ninguna tupla de la relación del lado derecho se completan con valores nulos y se añaden al resultado. De forma análoga, las tuplas de la relación del lado derecho que no encajen con ninguna tupla de la relación del lado izquierdo, se completan con valores nulos y se añaden al resultado.

Por ejemplo, la Figura 4.7 muestra el resultado de la expresión

préstamo **full outer join** *prestatario* **using** (*número-préstamo*)

Otro ejemplo del uso de la operación reunión externa es la consulta: «Listar todos los clientes que poseen una cuenta pero no tienen un préstamo en el banco». Esta consulta se puede formular como sigue:

select *i-NC*
from (*impositor* **left outer join** *prestatario*
on *impositor. nombre-cliente*
= prestatario.nombre-cliente)
as *db*1 (*i-NC*, *número-cuenta*, *p-NC*, *número-préstamo*)
where *p-NC* **is** *null*

De forma análoga, la consulta «Listar todos los clientes que tienen o bien una cuenta o un préstamo en el banco (pero no ambos)» podría formularse usando el operador de reunión natural externa completa, del modo siguiente:

select *nombre-cliente*
from (*impositor* **natural full outer join** *prestatario*)
where *número-cuenta* **is** *null* **or** *número-préstamo* **is** *null*

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*
P-170	Centro	3.000	Santos
P-230	Moralzarzal	4.000	Gómez
P-155	*null*	*null*	López

FIGURA 4.6. Resultado de *préstamo* **natural right outer join** *prestatario*.

SQL-92 proporciona además otros dos tipos de reunión, llamados **cross join** (reunión cruzada) y **union join** (reunión de unión). El primero es equivalente a una reunión interna sin condición de reunión; el segundo es equivalente a una reunión externa completa con condición falsa, es decir, donde la reunión interna es vacía.

4.11. LENGUAJE DE DEFINICIÓN DE DATOS

En muchos de nuestros análisis sobre SQL y bases de datos relacionales hemos aceptado un conjunto de relaciones como predefinidas. Por supuesto, el conjunto de relaciones en una base de datos se debe especificar en términos de un lenguaje de definición de datos (LDD).

El LDD de SQL permite la especificación no sólo de un conjunto de relaciones, sino también de alguna información relativa a esas relaciones, incluyendo:

- El esquema de cada relación
- El dominio de valores asociado a cada atributo
- Las restricciones de integridad
- El conjunto de índices que se deben mantener por cada relación
- Información de seguridad y autorización para cada relación
- La estructura de almacenamiento físico de cada relación en disco.

Se analizará a continuación la definición de esquemas y de dominios de valores; el análisis de las demás características del LDD de SQL se realizará en el Capítulo 6.

4.11.1. Tipos de dominios en SQL

La norma SQL soporta un conjunto de tipos de dominios predefinidos, que incluye los siguientes:

- **char** (*n*) es una cadena de caracteres de longitud fija, con una longitud *n* especificada por el usuario. También se puede utilizar la palabra completa **character**.
- **varchar** (*n*) es una cadena de caracteres de longitud variable, con una longitud *n* especificada por el usuario. También se puede utilizar la forma completa **character varying.**.
- **int** es un entero (un subconjunto finito de los enteros, que es dependiente de la máquina). También se puede usar la palabra completa **integer**.
- **smallint** es un entero pequeño (un subconjunto del dominio de los enteros, también dependiente de la máquina).
- **numeric** (*p*,*d*) es un número en coma flotante, cuya precisión la especifica el usuario. El número está formado por *p* dígitos (más el signo), y de esos *p* dígitos, *d* pertenecen a la parte decimal. Así, **numeric** (3,1) permite que el número 44,5 se almacene exactamente, mientras que los números 444,5 y 0,32 no se pueden almacenar exactamente en un campo de este tipo.
- **real, double precision** son respectivamente números en coma flotante y números en coma flotante de doble precisión, con precisión dependiente de la máquina.
- **float** (*n*) es un número en coma flotante, cuya precisión es de al menos *n* dígitos.
- **date** es una fecha del calendario, que contiene un año (de cuatro dígitos), un mes y un día del mes.
- **time** es la hora del día, expresada en horas, minutos y segundos. Se puede usar una variante, **time**(*p*), para especificar el número de dígitos decimales para los segundos (el número predeterminado es 0). También es posible almacenar la información del uso horario junto al tiempo.
- **timestamp** es una combinación de **date** y **time**. Se puede usar una variante, **timestamp**(*p*), para

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*
P-170	Centro	3.000	Santos
P-230	Moralzarzal	4.000	Gómez
P-260	Navacerrada	1.700	*null*
P-155	*null*	*null*	López

FIGURA 4.7. Resultado de *préstamo* **full outer join** *prestatario* **using** (*número-préstamo*).

especificar el número de dígitos decimales para los segundos (el número predeterminado es 6).

Los valores de fecha y hora se pueden especificar como:

date '2001-04-25'
time '09:30:00'
timestamp '2001-04-25 10:29:01.45'

Las fechas se deben especificar en el formato año seguido de mes y de día, como se muestra. El campo segundos de **time** y **timestamp** puede tener parte decimal, como se ha mostrado. Se puede usar una expresión de la forma **cast** *e* **as** *t* para convertir una cadena de caracteres (o una expresión de tipo cadena) *e* al tipo *t*, donde *t* es **date**, **time** o **timestamp**. La cadena debe estar en el formato adecuado como se indicó al comienzo de este párrafo.

Para extraer campos individuales de un valor *d* **date** o **time** se puede usar **extract**(*campo* **from** *d*), donde *campo* puede ser **year**, **month**, **day**, **hour**, **minute** o **segundo**.

Las cadenas de caracteres de longitud variable, la fecha y la hora no forman parte de la norma SQL.

SQL permite realizar operaciones de comparación sobre todos los dominios que se listan aquí, y permite realizar operaciones aritméticas y de comparación sobre los diferentes dominios numéricos. SQL también proporciona un tipo de datos llamado **interval** y permite realizar cálculos basados en fechas, horas e intervalos. Por ejemplo, si *x* e *y* son del tipo **date**, entonces *x*-*y* será un intervalo cuyo valor es el número de días desde la fecha *x* hasta la *y*. De forma análoga, al sumar o restar un intervalo de una fecha u hora, se obtendrá como resultado otra fecha u hora, respectivamente.

A menudo es útil poder comparar valores de dominios **compatibles**. Por ejemplo, como cada entero perteneciente al tipo **smallint** es un entero, una comparación $x < y$, donde *x* es de tipo *smallint* e *y* es de tipo **int** (o viceversa), es válida. Este tipo de comparación se lleva a cabo transformando primero el número *x* en un entero. Una transformación de este tipo se denomina *coerción*. La **coerción de tipos** se usa normalmente en los lenguajes de programación comunes, así como en los sistemas de bases de datos.

Como ejemplo, supóngase que el dominio de *nombre-cliente* es una cadena de caracteres de longitud 20 y que el dominio de *nombre-sucursal* es una cadena de caracteres de longitud 15. Aunque las longitudes de las cadenas de caracteres difieran, la norma SQL los considerará tipos compatibles.

Como se estudió en el Capítulo 3, el valor *null* pertenece a todos los dominios. Para ciertos atributos, sin embargo, los valores nulos pueden no ser apropiados. Considérese una tupla de la relación *cliente* donde el *nombre-cliente* es nulo. Una tupla de este tipo asociaría una calle y una ciudad a clientes anónimos: es decir, no contendrán información útil. En casos de este tipo, sería deseable prohibir el uso de valores nulos, restringiendo el dominio de *nombre-cliente* para excluir los valores nulos.

SQL permite incluir en la declaración de dominio de un atributo la especificación **not null** y de este modo se prohíbe la inserción de un valor nulo para ese atributo. Cualquier modificación de la base de datos que conduzca a la inserción de un valor nulo en un dominio especificado como **not null**, generará un diagnóstico de error. Existen muchas situaciones en las que sería deseable la prohibición de valores nulos. Un caso particular donde es esencial prohibir valores nulos es en la clave primaria de un esquema de relación. De este modo, en el ejemplo bancario, en la relación *cliente* se prohibirá el uso de valores nulos para el atributo *nombre-cliente*, que es la clave primaria de la relación.

4.11.2. Definición de esquemas en SQL

Un esquema de relación se define utilizando la orden **create table**:

create table r (A_1D_1, A_2D_2, … A_nD_n,
⟨restricción-integridad$_1$⟩.
…
⟨restricción-integridad$_k$⟩)

donde r es el nombre de la relación, cada A_i es el nombre de un atributo del esquema de relación r y D_i es el dominio de los valores del atributo A_i. Las restricciones de integridad válidas incluyen:

- **primary key** (A_{j_1}, A_{j_2},…,A_{j_m}): la especificación de **clave primaria** dice que los atributos A_{j_1}, A_{j_2},…,A_{j_m} forman la clave primaria de la relación. Los atributos clave primaria deben ser *no nulos* y *únicos*; es decir, ninguna tupla puede tener un valor nulo para un atributo de la clave primaria y ningún par de tuplas de la relación pueden ser iguales en todos los atributos clave primaria[1]. Aunque la especificación de clave primaria es opcional, es generalmente buena idea especificar una clave primaria para cada relación.
- **check** (P): la cláusula **check** especifica un predicado P que debe satisfacer cada tupla de la relación.

La orden de creación de tabla **create table** también incluye otras restricciones de integridad, que se estudiarán en el Capítulo 6.

En la Figura 4.8 se representa una definición parcial en SQL de la base de datos bancaria. Obsérvese que, al igual que en capítulos anteriores, no se intenta modelar con precisión el mundo real en el ejemplo de la base de

[1] En SQL-89, los atributos que forman la clave primaria no estaban declarados implícitamente como **not null**; era necesario una declaración explícita.

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

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

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

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

FIGURA 4.8. Definición de datos en SQL para parte de la base de datos del banco.

datos bancaria. En el mundo real, muchas personas tienen el mismo nombre por lo que *nombre-cliente* no sería una clave primaria de cliente; probablemente se usaría un *id-cliente* como clave primaria. Se usa *nombre-cliente* como clave primaria para mantener el esquema de la base de datos simple y pequeño.

Si como resultado de una inserción o modificación, una tupla toma valores nulos para cualquiera de los atributos que forman parte de la clave primaria, o si tiene el mismo valor que otra tupla de la relación para éstos, SQL notifica el error y la actualización no se lleva a cabo. De forma análoga ocurre lo mismo si falla la condición **check** de una tupla.

De manera predeterminada, **null** es un valor válido para cualquier atributo en SQL, a menos que se especifique con **not null**. Un atributo se puede declarar para que no sea nulo de la forma siguiente.

número-cuenta **char**(10) **not null**

SQL también soporta una restricción de integridad

unique (A_{j_1}, A_{j_2},…,A_{j_m})

La especificación **unique** indica que los atributos A_{j_1}, A_{j_2},…,A_{j_m} forman una clave candidata; es decir, no puede haber dos tuplas en la relación con todos los atributos que forman la clave candidata iguales. Sin embargo, se permite que los atributos que forman la clave candidata sean nulos, a menos que se hayan declarado como **not null**. Recuérdese que un valor nulo no es igual a ningún otro valor. El tratamiento de los valores nulos aquí es el mismo que para la constructora **unique** definida en el Apartado 4.6.4.

Un uso habitual de la cláusula **check** es el de asegurar que los valores de los atributos satisfacen unas condiciones determinadas, constituyendo así un poderoso sistema de tipos. Por ejemplo, la cláusula **check** en la orden de creación de una tabla para la relación *sucursal* comprueba que el valor para el atributo *activo* es un entero positivo. Considérese el siguiente ejemplo:

```
create table estudiante
    (nombre           char (15) not null,
     id-estudiante    char (10) not null,
     nivel-estudios   char (15) not null,
     primary key (id-estudiante),
     check (nivel-estudios in ('Graduado', 'Licenciado',
            'Doctorado' )))
```

En este ejemplo se utiliza la cláusula **check** para simular un tipo enumerado especificando que *nivel-estudios* debe ser «Graduado», «Licenciado» o «Doctorado». En el Capítulo 6 se considerarán condiciones **check** más generales, así como clases de restricciones denominadas restricciones de integridad.

Una relación inicialmente está vacía. Se pueden utilizar instrucciones de inserción para introducir datos en la misma. Muchas bases de datos relacionales tienen utilidades de carga para la introducción de un conjunto inicial de tuplas en una relación.

Para borrar una relación de una base de datos SQL, se utiliza la orden **drop table**. Dicha orden borra de la base de datos toda la información sobre la relación eliminada. La instrucción

drop table *r*

tiene una repercusión más drástica que

delete from *r*

La última conserva la relación *r*, pero borra todas sus tuplas. La primera, no sólo borra todas las tuplas de la relación *r*, sino también borra su esquema. Después de que *r* se elimine no se puede insertar ninguna tupla en dicha relación, a menos que su esquema se vuelva a crear utilizando la instrucción **create table**.

En SQL-92, la instrucción **alter table** se utiliza para añadir atributos a una relación existente. La sintaxis de la instrucción es la siguiente:

alter table *r* **add** *A D*

donde *r* es el nombre de una relación existente, *A* es el nombre del atributo que se desea añadir y *D* es el dominio del atributo *A*. Se pueden eliminar atributos de una relación utilizando la orden

alter table *r* **drop** *A*

donde *r* es el nombre de una relación existente y *A* es el nombre de un atributo de la relación. Muchos sistemas de bases de datos no permiten el borrado de atributos, aunque sí permiten el borrado de una tabla completa.

4.12. SQL INCORPORADO

SQL proporciona un lenguaje de consultas declarativo muy potente. La formulación de consultas en SQL es normalmente mucho más sencilla que la formulación de las mismas en un lenguaje de programación de propósito general. Sin embargo, el acceso a una base de datos desde un lenguaje de programación de propósito general se hace necesario al menos por dos razones:

1. No todas las consultas pueden expresarse en SQL, ya que SQL no dispone del poder expresivo de un lenguaje de propósito general. Así, existen consultas que se pueden expresar en lenguajes como Pascal, C, Cobol o Fortran y que no se pueden expresar en SQL. Para formular consultas de este tipo, podemos utilizar SQL dentro de un lenguaje más potente.

   SQL está diseñado de tal forma que las consultas formuladas puedan optimizarse automáticamente y ejecutarse de manera eficiente (al proporcionar toda la potencia de un lenguaje de programación, la optimización automática es extremadamente difícil.
2. Las acciones no declarativas (como la impresión de un informe, la interacción con un usuario o el envío de los resultados de una consulta a una interfaz gráfica) no se pueden llevar a cabo desde el propio SQL. Normalmente las aplicaciones tienen varios componentes y la consulta o actualización de datos es uno de ellos; los demás componentes se escriben en lenguajes de programación de alto nivel. En el caso de una aplicación integrada, los programas escritos en el lenguaje de programación deben tener la capacidad de acceder a la base de datos.

La norma SQL define la utilización de SQL dentro de varios lenguajes de programación, tales como C, Cobol, Pascal, Java, PL/I y Fortran. Un lenguaje en el cual se introducen consultas SQL se denomina lenguaje *anfitrión* y las estructuras SQL que se admiten en el lenguaje anfitrión constituyen SQL *incorporado*.

Los programas escritos en el lenguaje anfitrión pueden usar sintaxis SQL para acceder y actualizar datos almacenados en la base de datos. Esta forma incorporada de SQL amplía aún más la capacidad de los programadores de manipular la base de datos. En SQL incorporado, la ejecución de una consulta la realiza el sistema de base de datos y el resultado de la misma se hace disponible al programa, tupla a tupla (registro).

Un programa con SQL incorporado debe tratarse con un preprocesador especial antes de la compilación. Las consultas de SQL incorporado se sustituyen por declaraciones escritas en el lenguaje anfitrión y por llamadas a procedimientos que permiten la ejecución del acceso a la base de datos. Tras esta operación, el programa resultado se compila con el compilador del lenguaje anfitrión. Para identificar las consultas de SQL incorporado, se utiliza la instrucción EXEC SQL, que tiene la siguiente forma:

```
EXEC SQL <instrucción de SQL incorporado>
  END-EXEC
```

La sintaxis exacta de las consultas en SQL incorporado depende del lenguaje dentro del que se utilicen. Por ejemplo, cuando se utilizan instrucciones de SQL dentro de un programa en C, se debe utilizar un punto y coma en lugar de END-EXEC. La incorporación de SQL en Java (denominada SQLJ) usa la sintaxis

```
# SQL { <instrucción de SQL incorporado> };
```

En el programa se incluye la instrucción SQL INCLUDE para identificar el lugar donde el preprocesador debe insertar las variables especiales que se usan para la comunicación entre el programa y el sistema de base de datos. Las variables del lenguaje anfitrión se pueden utilizar en las instrucciones de SQL incorporado, pero se precederán por dos puntos (:) para distinguirlas de las variables de SQL.

Las instrucciones de SQL incorporado son similares en cuanto a la sintaxis a las instrucciones SQL que se han descrito en este capítulo. Sin embargo, hay varias diferencias que se indican a continuación.

Para formular una consulta relacional se usa la instrucción **declare cursor**. El resultado de la consulta no se calcula aún. En lugar de esto, el programa debe usar las órdenes **open** y **fetch** (que se analizarán más adelante en este apartado) para obtener las tuplas resultado.

Considerando el esquema bancario que se ha utilizado como ejemplo en este capítulo, supóngase que se tiene una variable del lenguaje anfitrión *importe* y que se desea encontrar los nombres y ciudades de residencia de aquellos clientes que superan esa cantidad en alguna de sus cuentas. Se puede escribir esta consulta del modo siguiente:

```
EXEC SQL
        declare c cursor for
        select nombre-cliente, ciudad-cliente
        from impositor, cliente
        where impositor.nombre-cliente
                = cliente.nombre-cliente and
                cuenta.número-cuenta
                = impositor.número-cuenta and
                impositor.saldo > :importe
END-EXEC
```

En la consulta anterior, la variable *c* se denomina *cursor* de la consulta. Se utiliza esta variable para identificar la consulta en la instrucción **open**, que ocasiona que se evalúe la consulta, y en la instrucción **fetch**, que permite que los valores de una tupla se obtengan como variables del lenguaje anfitrión.

La instrucción **open** para el ejemplo anterior debería ser:

EXEC SQL **open** *c* END-EXEC

Esta instrucción hace que el sistema de base de datos ejecute la consulta y guarde el resultado dentro de una relación temporal. La consulta tiene una variable del lenguaje anfitrión (:*importe*); la consulta usa el valor de la variable en el momento en que se ejecuta la instrucción **open**.

Si se produce un error como resultado de la consulta SQL, el sistema de base de datos almacena un diagnóstico de error en las variables del área de comunicación SQL (SQLCA), cuya declaración se hace mediante la instrucción SQL INCLUDE.

Un programa de SQL incorporado ejecuta una serie de instrucciones **fetch** para obtener las tuplas del resultado. La instrucción **fetch** necesita una variable del lenguaje anfitrión por cada atributo de la relación resultado. En el ejemplo anterior se necesita una variable para almacenar el valor de *nombre-cliente* y otra para el valor de *ciudad-cliente*. Si dichas variables fuesen *nc* y *cc* respectivamente, una tupla de la relación resultado se obtendría mediante la instrucción:

EXEC SQL **fetch** *c* **into** :*nc*, :*cc* END EXEC

A partir de este momento el programa puede manipular las variables *nc* y *cc*, utilizando las facilidades proporcionadas por el lenguaje anfitrión.

Un único **fetch** devuelve únicamente una tupla. Si se desean obtener todas las tuplas del resultado, el programa debe tener un bucle para iterar sobre todas las tuplas. SQL incorporado ofrece una ayuda para el programador a la hora de programar esta iteración. Aunque una relación es conceptualmente un conjunto, las tuplas resultado de una consulta están físicamente en un determinado orden fijo. Cuando se ejecuta una instrucción **open**, el cursor pasa a apuntar a la primera tupla del resultado. Al ejecutarse una instrucción **fetch**, el cursor se actualiza, pasando a apuntar a la siguiente tupla del resultado. Cuando no quedan más tuplas para ser procesadas, la variable SQLSTATE en SQLCA se establece a '02000' (que significa «sin datos»). Una variable de SQLCA indica que ya no quedan tuplas por analizar en el resultado. Así, se puede utilizar un bucle **while** (o el equivalente, en función del lenguaje anfitrión) para procesar cada tupla del resultado.

La instrucción **close** se debe utilizar para indicar al sistema de base de datos que borre la relación temporal que contenía el resultado de la consulta. Para el ejemplo anterior, la sintaxis de esta instrucción será:

EXEC SQL **close** *c* END-EXEC

Las expresiones de SQL incorporado que se utilizan para modificaciones (actualización, inserción y borrado) de bases de datos no devuelven ningún resultado. Además, son más simples de expresar. Una instrucción de modificación tiene el siguiente aspecto:

EXEC SQL <cualquier actualización, inserción
o borrado válidos> END-EXEC

En la expresión de modificación pueden aparecer variables del lenguaje anfitrión precedidas de dos puntos. Si se produce un error en la ejecución de la instrucción, se devuelve un diagnóstico en SQLCA.

Las relaciones de la base de datos también se pueden actualizar con cursores. Por ejemplo, si se desea añadir 100 al atributo *saldo* de cada *cuenta* donde el nombre de sucursal sea «Navacerrada», se podría declarar un cursor como:

```
declare c cursor for
select *
from account
where nombre-sucursal = 'Navacerrada'
for update
```

Se puede iterar por las tuplas ejecutando operaciones **fetch** sobre el cursor (como se mostró anteriormente) y después de obtener cada tupla se ejecuta el siguiente código:

```
update cuenta
set saldo = saldo +100
where current of c
```

SQL incorporado permite a un programa en el lenguaje anfitrión acceder a la base de datos, pero no proporciona ayuda para presentar los resultados al usuario o al generar informes. La mayoría de productos comerciales de bases de datos incluyen herramientas para ayudar a los programadores de aplicaciones a crear interfaces de usuario e informes con formato. Estas herramientas se estudian en el Capítulo 5 (Apartado 5.3).

4.13. SQL DINÁMICO

El componente *dinámico* de SQL-92 permite que en un programa se construyan y realicen consultas SQL en tiempo de ejecución. En cambio, las instrucciones de SQL incorporado deben estar presentes en tiempo de compilación y se compilan utilizando un preprocesador de SQL incorporado. Por medio de SQL dinámico los programas pueden crear consultas SQL en tiempo de ejecución (tal vez basadas en datos introducidos por el usuario) y pueden ejecutarlas inmediatamente o dejarlas *preparadas* para su ejecución. La preparación de una instrucción SQL dinámica la compila y los usos posteriores de la instrucción preparada usan la versión compilada.

SQL define normas para incorporar las llamadas de SQL dinámico dentro de lenguaje anfitrión, como C, como se muestra en el siguiente ejemplo.

```
char * prog_sql =«update cuenta set saldo
                = saldo * 1.05
                where número-cuenta = ?»
EXEC SQL prepare prog_din from :prog_sql;
char cuenta[10] = «C-101»;
EXEC SQL execute prog_din using :cuenta;
```

El programa de SQL dinámico contiene una interrogación '?' que representa una variable que se debe proporcionar en la ejecución del programa.

Sin embargo, esta sintaxis requiere extensiones para el lenguaje o un preprocesador para el lenguaje extendido. Una alternativa que usa ampliamente es una interfaz para programas de aplicación para enviar las consultas SQL o actualizaciones a un sistema de bases de datos, sin realizar cambios en el propio lenguaje de programación.

En el resto de este apartado se examinan dos normas de conexión a una base de datos SQL y la realización de consultas y actualizaciones. Una, ODBC, es una interfaz para programas de aplicación para el lenguaje C, mientras que la otra es para Java.

Para comprender estas normas es necesario comprender el concepto de sesión SQL. El usuario o aplicación se *conecta* a un servidor SQL, estableciendo una sesión; ejecuta una serie de instrucciones y, finalmente, *desconecta* la sesión. Así, todas las actividades del usuario o aplicación están en el contexto de una sesión SQL. Además de las órdenes normales de ÇSQL, una sesión también puede contener órdenes para *comprometer* el trabajo realizado en la sesión o para *retrocederlo*.

4.13.1. ODBC**

La norma **ODBC** (Open Database Connectivity, conectividad abierta de bases de datos) define una forma para que un programa de aplicación se comunique con un servidor de bases de datos. ODBC define una **interfaz para programas de aplicación** (API, Application Program Interface) que pueden usar las aplicaciones para abrir una conexión con una base de datos, enviar consultas y actualizaciones y obtener los resultados. Las aplicaciones como las interfaces gráficas de usuario, los paquetes estadísticos y las hojas de cálculo pueden usar la misma API ODBD para conectarse a cualquier servidor de bases de datos compatible con ODBC.

Cada sistema de bases de datos que sea compatible con ODBC proporciona una biblioteca que se debe enlazar con el programa cliente. Cuando el programa cliente realiza una llamada a la API ODBC, el código de la biblioteca se comunica con el servidor para realizar la acción solicitada y obtener los resultados.

La Figura 4.9 muestra un ejemplo de código C que usa la API ODBC. El primer paso al usar ODBC para comunicarse con un servidor es configurar la conexión con el servidor. Para ello, el programa asigna en primer lugar un entorno SQL, después un manejador para la conexión a la base de datos. ODBC define los tipos HENV, HDBC y RETCODE. El programa abre a continuación la conexión a la base de datos usando SQLConnect. Esta llamada tiene varios parámetros, incluyendo el manejador de la conexión, el servidor al que conectarse, el identificador de usuario y la contraseña. La constante SQL_NTS denota que el argumento anterior es una cadena terminada con nulo.

Una vez que se ha configurado la conexión, el programa puede enviar órdenes SQL a la base de datos usando SQLExecDirect. Las variables del lenguaje C se pueden vincular a los atributos del resultado de la consulta, de forma que cuando se obtenga una tupla resultado usando SQLFetch, sus valores de atributo se almacenan en las variables C correspondientes. La función SQLBindCol realiza esta tarea; el segundo argumento identifica la posición del atributo en el resultado de la consulta, y el tercer argumento indica la conversión de tipos de SQL a C requerida. El siguiente argumento da la dirección de la variable. Para los tipos de longitud variables como los arrays de caracteres, los dos últimos argumentos dan la longitud máxima de la variable y una ubicación donde almacenar la longitud actual cuando se obtenga una tupla. Un valor negativo devuelto para el campo longitud indica que el valor es **null**.

La instrucción SQLFetch está en un bucle **while** que se ejecuta hasta que SQLFetch devuelva un valor diferente de SQL_SUCCESS. En cada obtención de valores, el programa los almacena en variables C como se especifica en las llamadas en SQLBindCol e imprime estos valores.

Al final de la sesión, el programa libera el manejador, se desconecta de la base de datos y libera la conexión y los manejadores del entorno SQL. Un buen estilo de programación requiere que el resultado de cada

```
int ODBCexample()
{
    RETCODE error;
    HENV ent; /* entorno */
    HDBC con; /* conexión a la base de datos */

    SQLAllocEnv(&ent);
    SQLAllocConnect(ent, &con);
    SQLConnect(con, «aura.bell-labs.com», SQL NTS, «avi», SQL NTS, «avipasswd», SQL NTS);
    {
        char nombresucursal[80];
        float saldo;
        int lenOut1, lenOut2;
        HSTMT stmt;

        char * consulta = «select nombre_sucursal, sum (saldo)
                           from cuenta
                           group by nombre_sucursal»;
        SQLAllocStmt(con, &stmt);
        error = SQLExecDirect(stmt, consulta, SQL NTS);
        if (error == SQL SUCCESS) {
            SQLBindCol(stmt, 1, SQL C CHAR, nombresucursal, 80, &lenOut1);
            SQLBindCol(stmt, 2, SQL C FLOAT, &saldo, 0 , &lenOut2);
            while (SQLFetch(stmt) >= SQL SUCCESS) {
                printf (« %s %g\n», nombresucursal, saldo);
            }
        }
        SQLFreeStmt(stmt, SQL DROP);
    }
    SQLDisconnect(con);
    SQLFreeConnect(con);
    SQLFreeEnv(ent);
}
```

FIGURA 4.9. Código de ejemplo ODBC.

función se comprueba para asegurarse de que no haya errores; se han omitido la mayoría de estas comprobaciones por brevedad.

Es posible crear una instrucción SQL con parámetros; por ejemplo, considérese la instrucción insert into account values(?,?,?). Los interrogantes son resguardos para los valores que se proporcionarán después. Esta instrucción se puede «preparar», es decir, compilar en la base de datos y ejecutar repetidamente proporcionando los valores reales para los resguardos —en este caso proporcionando un número de cuenta, nombre de sucursal y saldo para la relación *cuenta*.

ODBC define funciones para varias tareas, tales como hallar todas las relaciones en la base de datos y los nombres y tipos de las columnas del resultado de una consulta o una relación de la base de datos.

De forma predeterminada, cada instrucción SQL se trata como una transacción separada que se compromete automáticamente. La llamada SQLSetConnectOption(con, SQL_AUTOCOMMIT, 0) desactiva el compromiso automático en la conexión con, y las transacciones se deben comprometer explícitamente con SQLTransact(con, SQL_COMMIT) o retroceder con SQLTransact(con, SQL_ROLLBACK).

Las versiones más recientes de la norma ODBC añaden nueva funcionalidad. Cada versión define *niveles de acuerdo* que especifican subconjuntos de la funcionalidad definida por el estándar. Una implementación ODBC puede proporcionar sólo las características básicas o puede proporcionar características más avanzadas (nivel 1 o 2). El nivel 1 requiere soporte para la obtención de información del catálogo, como la información sobre las relaciones existentes y los tipos de sus atributos. El nivel 2 requiere más características, como la capacidad de enviar y obtener arrays de valores de parámetros y para obtener información del catálogo más detallada.

Las normas más recientes de SQL (SQL-92 y SQL:1999) definen una **interfaz en el nivel de llamada** (Call-Level Interface, CLI) que es similar a la interfaz ODBC, pero con algunas pequeñas diferencias.

4.13.2. JDBC**

La norma **JDBC** define una API que pueden usar los programas Java para conectarse a los servidores de bases de datos (la palabra JDBC fue originalmente abreviatura de «Java Database Connectivity» —conectividad de bases de datos con Java— pero la forma completa ya no se usa). La Figura 4.10 muestra un ejemplo de un programa Java que usa la interfaz JDBC. El programa debe en primer lugar abrir una conexión a una base de datos y después ejecutar instrucciones SQL, pero antes de abrir una conexión, carga los controladores adecuados para la base de datos usando Class.forName. El primer parámetro de la llamada getConnection especifica el

```
public static void ejemploJDBC (String idbd, String idusuario, String contraseña)
{
    try
    {
        Class.forName («oracle.jdbc.driver.OracleDriver»);
        Connection con = DriverManager.getConnection
                    («jdbc:oracle:thin:@aura.bell-labs.com:2000:bdbanco»,
                    idusuario, contraseña);
        Statement stmt = con.createStatement();
        try {
            stmt.executeUpdate(
                    «insert into cuenta values('C-9732', 'Navacerrada', 1200)»);
        } catch (SQLException sqle)
        {
            System.out.println(«No se pudo insertar la tupla. » + sqle);
        }
        ResultSet rset = stmt.executeQuery
                    («select nombre_sucursal, avg (saldo)
                    from cuenta
                    group by nombre_sucursal»);
        while (rset.next()) {
            System.out.println(rset.getString(«nombre_sucursal») + « » +
                        rset.getFloat(2));
        }
        stmt.close();
        con.close();
    }
    catch (SQLException sqle)
    {
        System.out.println(«SQLException : » + sqle);
    }
}
```

FIGURA 4.10. Un ejemplo de código JDBC.

nombre de la máquina en la que se ejecuta el servidor (en este caso, aura.bell-labs.com) y el número de puerto que usa para la comunicación (en este caso, 2000). El parámetro también especifica el esquema de la base de datos a usar (en este caso, bdbanco), ya que un servidor de bases de datos puede dar soporte a varios esquemas. El primer parámetro también especifica el protocolo a usar para la comunicación con la base de datos (en este caso, jdbc:oracle:thin:). Obsérvese que JDBC especifica sólo la API, no el protocolo de comunicación. Un controlador JDBC puede dar soporte a varios protocolos y se debe especificar el compatible con la base de datos y el controlador. Los otros dos argumentos de getConnection son un identificador de usuario y una contraseña.

El programa crea a continuación un manejador para la conexión y lo usa para ejecutar una instrucción SQL y obtener los resultados. En nuestro ejemplo, stmt.executeUpdate ejecuta una instrucción de actualización. El constructor try {…} catch {…} permite capturar cualquier excepción (condición de error) que surjan cuando se realizan las llamadas JDBC, e imprime un mensaje apropiado para el usuario.

El programa puede ejecutar una consulta usando stmt.executeQuery. Puede obtener el conjunto de filas en el resultado en ResultSet y leer tupla a tupla usando la función next() en el conjunto de resultados. La Figura 4.10 muestra dos formas de obtener los valores de los atributos en una tupla: usando el nombre del atributo (*nombre-sucursal*) y usando la posición del atributo (2, para denotar el segundo atributo).

También se puede crear una instrucción preparada en la que algunos valores se reemplacen por «?», especificando que los valores actuales se proporcionarán más tarde. Se pueden proporcionar los valores usando setString(). La base de datos puede compilar la consulta cuando esté preparada, y cada vez que se ejecute (con nuevos valores), la base de datos puede rehusar la forma compilada previamente de la consulta. El fragmento de código de la Figura 4.11 muestra cómo se pueden usar las instrucciones preparadas.

JDBC proporciona otras características, como los **conjuntos de resultados actualizables**. Puede crear un conjunto de resultados actualizable a partir de una consulta que realice una selección o una proyección de una

```
PreparedStatement pStmt = con.prepareStatement(
            «insert into cuenta values(?,?,?)»);
pStmt.setString(1, «C-9732»);
pStmt.setString(2, «Navacerrada»);
pStmt.setInt(3, 1200);
pStmt.executeUpdate();
pStmt.setString(1, «C-9733»);
pStmt.executeUpdate();
```

FIGURA 4.11. Instrucciones preparadas en código JDBC.

relación de la base de datos. Una actualización de una tupla en el conjunto de resultados es consecuencia de una actualización de la tupla correspondiente de la relación de la base de datos. JDBC también proporciona una API para examinar esquemas de la base de datos para encontrar los tipos de atributos de un conjunto de resultados.

Para obtener más información sobre JDBC, consúltese la información bibliográfica al final del capítulo.

4.14. OTRAS CARACTERÍSTICAS DE SQL**

El lenguaje SQL ha crecido durante las dos décadas pasadas desde un lenguaje simple con pocas características a un lenguaje ciertamente complejo con características para satisfacer a muchos tipos diferentes de usuarios. Se trataron los fundamentos de SQL anteriormente en este capítulo. En este apartado se introducen al lector algunas de las características más complejas de SQL.

4.14.1. Esquemas, catálogos y entornos

Para comprender la motivación de los esquemas y los catálogos, considérese cómo se denominan los archivos en un sistema de archivos. Los sistemas de archivos originales eran planos; es decir, todos los archivos se almacenaban en un directorio. Los sistemas de archivos de la generación actual tienen por supuesto una estructura de directorios, con archivos almacenados en subdirectorios. Para denominar unívocamente un archivo se debe especificar el nombre completo de la ruta del archivo, por ejemplo /usuarios/avi/db-book/capítulo4.tex.

Al igual que en los primeros sistemas de archivos, los primeros sistemas de bases de datos tenían un único espacio de nombres para todas las relaciones. Los usuarios tenían que coordinarse para asegurarse de que no intentaban usar el mismo nombre para relaciones diferentes. Los sistemas de bases de datos actuales proporcionan una jerarquía de tres niveles para denominar a las relaciones. El nivel superior de la jerarquía consiste en **catálogos**, cada uno de los cuales puede contener **esquemas**. Los objetos SQL tales como las relaciones y las vistas están contenidos en un **esquema**.

Para realizar cualquier acción sobre una base de datos, un usuario (o un programa) debe en primer lugar *conectarse* a la base de datos. El usuario debe proporcionar el nombre de usuario y generalmente una contraseña secreta para comprobar la identidad del usuario, como se vio en los ejemplos de ODBC y JDBC de los apartados 4.13.1 y 4.13.2. Cada usuario tiene un catálogo y esquema predeterminados, y la combinación es única para el usuario. Cuando un usuario se conecta a un sistema de bases de datos, el catálogo y esquema predeterminados se configuran para la conexión; esto se corresponde con el directorio actual establecido para el directorio inicial del usuario cuando el usuario inicia la sesión en el sistema operativo.

Para identificar una relación unívocamente, se debe usar un nombre con tres partes, por ejemplo:

catálogo5.esquema-banco.cuenta

Se puede omitir el componente catálogo y, en ese caso, la parte catálogo del nombre se considera el catálogo predeterminado para la conexión. Así, si catálogo5 es el catálogo predeterminado, se puede usar esquema-banco.cuenta para identificar la misma relación unívocamente. Además, también se puede omitir el nombre del esquema, y la parte esquema del nombre es de nuevo considerada como el esquema predeterminado de la conexión. Así, se puede usar tan solo cuenta si el catálogo predeterminado es catálogo5 y el esquema predeterminado es esquema-banco.

Con varios catálogos y esquemas disponibles pueden trabajar independientemente diferentes aplicaciones y usuarios sin preocuparse acerca de la coincidencia de nombres. Además, pueden ejecutarse varias versiones de una aplicación (una versión de producción y otras de test) en el mismo sistema de bases de datos.

El catálogo y esquema predeterminados son parte de un **entorno SQL** que se configura por cada conexión. El entorno también contiene el identificador de usuario (también conocido como *identificador de autorización*). Todas las consultas SQL habituales, incluyendo las instrucciones LDD y LMD operan en el contexto de un esquema. Los esquemas se pueden crear o eliminar mediante las instrucciones **create schema** o **drop schema**. La creación y borrado de catálogos es dependiente de la implementación y no es parte de la norma SQL.

4.14.2. Extensiones procedimentales y procedimientos almacenados

SQL proporciona un lenguaje de **módulos**, que permite definir los procedimientos en SQL. Un módulo contiene normalmente varios procedimientos SQL. Cada procedimiento tiene un nombre, parámetros opcionales y una instrucción SQL. Una extensión del lenguaje estándar SQL-92 también permite constructoras procedimentales, tales como **for**, **while** e **if-then-else**, e instrucciones SQL compuestas (varias instrucciones SQL entre **begin** y **end**).

Los procedimientos se pueden almacenar en la base de datos y ejecutarse con la instrucción **call**. Estos procedimientos se denominan también **procedimientos almacenados**. Los procedimientos almacenados son particularmente útiles porque permiten que las opera-

ciones de la base de datos se encuentren disponibles a aplicaciones externas, sin exponer ninguno de los detalles internos de la base de datos.

El Capítulo 9 trata las extensiones procedimentales de SQL, así como otras muchas características nuevas de SQL:1999.

4.15. RESUMEN

- Los sistemas de bases de datos comerciales no utilizan los lenguajes de consulta formales descritos en el Capítulo 3. El ampliamente usado lenguaje SQL, que se ha estudiado en este capítulo, está basado en el álgebra relacional formal, pero con mucho «azúcar sintáctico».
- SQL incluye varias constructoras del lenguaje para las consultas sobre la base de datos. Todas las operaciones del álgebra relacional, incluyendo las operaciones del álgebra relacional extendida, se pueden expresar en SQL. SQL también permite la ordenación de los resultados de una consulta en términos de los atributos.
- Las relaciones de vistas se pueden definir como relaciones que contienen el resultado de consultas. Las vistas son útiles para ocultar información innecesaria y para recolectar información de más de una relación en una única vista.
- Las vistas temporales definidas con la cláusula **with** también son útiles para descomponer consultas complejas en partes más pequeñas y fáciles de entender.
- SQL incluye constructoras para insertar, actualizar y borrar información. Una transacción consiste en una secuencia de operaciones que deben ser atómicas. Es decir, todas las operaciones se realizan con éxito o ninguna. En la práctica, si una transacción no se puede completar con éxito, todas las acciones parciales realizadas se deshacen.
- Las modificaciones sobre la base de datos pueden conducir a la generación de valores nulos en las tuplas. Se estudió cómo se podían introducir los valores nulos y la forma en que SQL maneja las consultas sobre las relaciones que contienen estos valores.
- El lenguaje de definición de datos SQL se usa para crear relaciones con los esquemas especificados. El LDD de SQL soporta varios tipos incluyendo **date** y **time**. Más detalles del LDD de SQL, y en particular su soporte de las restricciones de integridad, aparecen en el Capítulo 6.
- Las consultas SQL se pueden llamar desde lenguajes anfitriones mediante SQL incorporado y dinámico. Las normas ODBC y JDBC definen interfaces para programas de aplicación para acceder a bases de datos SQL desde los programas en lenguaje C y Java. Los programadores usan cada vez más estas API para acceder a bases de datos.
- También se vio una visión general de algunas características avanzadas de SQL, tales como las extensiones procedimentales, los catálogos, los esquemas y los procedimientos almacenados.

TÉRMINOS DE REPASO

- Atomicidad
- Catálogo
- Cláusula **as**
- Cláusula **from**
- Cláusula **order by**
- Cláusula **select**
- Cláusula **where**
- Cláusula **with**
- Dominios
- Duplicados
- Esquema
- Funciones de agregación
  - **avg, min, max, sum, count**
  - **count**
- Índice
- JDBC
- LDD: lenguaje de definición de datos
- LMD: lenguaje de manipulación de datos
- Modificación de la base de datos
  - Actualización de vistas
  - **delete, insert, update**
- ODBC
- Operaciones de conjuntos
  - {<,<=,>,>=} {**some,all**}
  - **exists**
  - **unique**
- Operaciones de conjuntos
  - **union, intersect, except**

- Procedimientos almacenados
- Relaciones derivadas (en la cláusula **from**)
- SQL dinámico
- SQL incorporado
- Subconsultas anidadas
- Tipos de reunión
  - — **natural, using, on**
  - — Reunión externa por la izquierda, por la derecha y completa
  - — Reunión interna y externa
- Transacción
- Variable tupla
- Valores nulos
  - — Valor «desconocido»
- Vistas

EJERCICIOS

4.1. Considérese la base de datos de seguros de la Figura 4.12, donde las claves primarias se han subrayado. Formúlense las siguientes consultas SQL para esta base de datos relacional:

a. Buscar el número total de las personas cuyos coches se han visto involucrados en un accidente en 1989.
b. Buscar el número de accidentes en los cuales se ha visto involucrado un coche perteneciente a «Santos».
c. Añadir un nuevo accidente a la base de datos; supóngase cualquier valor para los atributos necesarios.
d. Borrar el Mazda de «Santos».
e. Actualizar el importe de daños del coche de matrícula «2002BCD» en el accidente con número de informe «AR2197» a 3.000 €.

4.2. Considérese la base de datos de empleados de la Figura 4.13, donde las claves primarias se han subrayado. Proporciónese una expresión SQL para cada una de las consultas siguientes:

a. Buscar los nombres de todos los empleados que trabajan en el Banco Importante.
b. Buscar los nombres y ciudades de residencia de todos los empleados que trabajan en el Banco Importante.
c. Buscar los nombres, direcciones y ciudades de residencia de todos los empleados que trabajan en el Banco Importante y que ganan más de 10.000 €.
d. Buscar todos los empleados que viven en la ciudad de la empresa para la que trabajan.
e. Buscar todos los empleados que viven en la misma ciudad y en la misma calle que sus jefes.
f. Buscar todos los empleados que no trabajan en el Banco Importante.
g. Buscar todos los empleados que ganan más que cualquier empleado del Banco Pequeño.
h. Supóngase que las empresas pueden tener sede en varias ciudades. Buscar todas las empresas con sede en todas las ciudades en las que tiene sede el Banco Pequeño.
i. Buscar todos los empleados que ganan más que el sueldo medio de los empleados de su empresa.
j. Buscar la empresa que tiene el mayor número de empleados.
k. Buscar la empresa que tiene el menor sueldo medio.
l. Buscar aquellas empresas cuyos empleados ganan un sueldo más alto, en media, que el sueldo medio del Banco Importante.

persona (id-conductor, nombre, dirección)
coche (matrícula, año, modelo)
accidente (número-informe, fecha, lugar)
es-dueño (id-conductor, matrícula)
participó (id-conductor, coche, número-informe, importe-daños)

FIGURA 4.12. Base de datos de seguros.

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

FIGURA 4.13. Base de datos de empleados.

4.3. Considérese la base de datos relacional de la Figura 4.13. Formúlese una expresión en SQL para cada una de las siguientes consultas:

a. Modificar la base de datos de forma que Santos viva en Ávila.
b. Incrementar en un 10% el sueldo de todos los empleados del Banco Importante.
c. Incrementar en un 10% el sueldo de todos los jefes del Banco Importante.
d. Incrementar en un 10% el sueldo de todos los empleados del Banco Importante, a menos que su sueldo pase a ser mayor de 100.000 €, en cuyo caso se incrementará su sueldo sólo en un 3%.
e. Borrar todas las tuplas de la relación *trabaja* correspondientes a los empleados del Banco Importante.

4.4. Considérense los esquemas de relación siguientes:

$$R = (A, B, C)$$
$$S = (D, E, F)$$

Además, considérense las relaciones $r(R)$ y $r(S)$. Obténgase la expresión SQL equivalente a las siguientes consultas:

a. $\Pi_A(r)$
b. $\sigma_{B=17}(r)$

c. $r \times s$
d. $\Pi_{A,F}(\sigma_{C=D}(r \times s))$

4.5. Sea $R = (A,B,C)$ y sean r_1 y r_2 relaciones sobre el esquema R. Proporciónese una expresión SQL equivalente a cada una de las siguientes consultas:

a. $r_1 \cup r_2$
b. $r_1 \cap r_2$
c. $r_1 - r_2$
d. $\Pi_{AB}(r_1) \bowtie \Pi_{BC}(r_2)$

4.6. Sea $R = (A,B)$ y $S=(A,C)$ y sean $r(R)$ y $s(S)$ relaciones. Formúlese una expresión SQL equivalente a cada una de las siguientes consultas:

a. $\{< a > | \exists\, b\ (< a, b > \in r \wedge b = 17)\}$
b. $\{< a, b, c > | < a, b > \in r \wedge < a, c > \in s\}$
c. $\{< a > | \exists\, c\ (< a, c > \in s \wedge \exists\, b_1, b_2\ (< a, b_1 > \in r \wedge < c, b_2 > \in r \wedge b_1 > b_2))\}$

4.7. Demuéstrese que en SQL **< > all** es equivalente a **not in**.

4.8. Considérese la base de datos relacional de la Figura 4.13. Utilizando SQL, defínase una vista que contenga *nombre-jefe* y el sueldo medio de todos los empleados que trabajan para ese jefe. Explíquese por qué el sistema de base de datos no debería permitir que las actualizaciones se expresaran en términos de esta vista.

4.9. Considérese la consulta SQL

select *p.a*1
from *p*, *r*1, *r*2
where *p.a*1 = *r*1.*a*1 **or** *p.a*1 = *r*2.*a*1

¿Bajo qué condiciones la consulta anterior devuelve los valores de *p.a*1 que están tanto en *r*1 como en *r*2? Examínense cuidadosamente los casos en los que *r*1.*a*1 o *r*2.*a*2 pueden ser nulos.

4.10. Escríbase una consulta SQL, sin usar la cláusula **with**, para encontrar todas las sucursales donde el depósito total de las cuentas sea menor que la media del depósito total medio en todas las sucursales usando:

a. Una consulta anidada en la cláusula **from**.
b. Una consulta anidada en una cláusula **having**.

4.11. Supóngase que se tiene una relación *nota (estudiante*, *puntuación)* y que se quiere clasificar a los estudiantes en función de la puntuación del modo siguiente:

SS: si la puntuación es menor que 5
AP: si la puntuación es mayor o igual que 5 y menor que 7
NT: si la puntuación es mayor o igual que 7 y menor que 8,5
SB: si la puntuación es mayor o igual que 8,5

Escríbanse consultas para hacer lo siguiente:

a. Mostrar la clasificación de cada estudiante, en términos de la relación *nota*.
b. Encontrar el número de estudiantes por clasificación.

4.12. SQL-92 proporciona una operación *n*-aria denominada **coalesce** que se define del modo siguiente: **coalesce** $(A_1, A_2, \ldots, A_n)$ devuelve el primer A_i no nulo en la lista $A_1, A_2, \ldots, A_n$ y devuelve nulo si todos ellos son nulos. Muéstrese cómo expresar la operación **coalesce** usando la operación **case**.

4.13. Sean *a* y *b* relaciones con los esquemas *A* (*nombre*, *dirección*, *puesto*) y *B* (*nombre*, *dirección*, *sueldo*), respectivamente. Indíquese cómo expresar *a* **natural full outer join** *b*, utilizando la operación **full outer join** con una condición **on** y la operación **coalesce**. Compruébese que la relación resultado no contiene dos copias de los atributos *nombre* y *dirección* y que la solución es válida incluso si dichos atributos de alguna tupla en *a* o *b* toman valor nulo para los atributos *nombre* o *dirección*.

4.14. Dada una definición de esquema SQL para la base de datos de empleados de la Figura 4.13, elíjase un dominio apropiado para cada atributo y una clave primaria para cada esquema de relación.

4.15. Escríbanse condiciones **check** para el esquema del ejercicio anterior para asegurar que:

a. Cada empleado trabaja para una empresa con sede en la ciudad de residencia del empleado.
b. Ningún empleado gana un sueldo mayor que el de su jefe.

4.16. Descríbanse las circunstancias bajo las cuales se debería utilizar SQL incorporado en lugar de SQL o un lenguaje de programación de propósito general.

NOTAS BIBLIOGRÁFICAS

Chamberlin et al. [1976] describen la versión original de SQL, denominada Sequel 2. Sequel 2 derivó de los lenguajes Square [Boyce et al., 1975] y Sequel [Chamberlin y Boyce, 1974]. La norma SQL-86 se describe en ANSI [1986]. IBM [1987] proporciona la definición de SQL de IBM System Application Architecture. Las normas oficiales de SQL-89 y de SQL-92 están disponibles en ANSI [1989] y ANSI [1992], respectivamente.

SQL-92 también se define en el U.S. Dept. of Commerce [1992] y en X/Open [1992, 1993]. Actualmente está en desarrollo la próxima versión de la norma de SQL, denominada SQL-3.

Algunos libros de texto que describen el lenguaje SQL-92 son Date y Darwen [1997], Melton y Simon [1993], y Cannan y Otten [1993]. Melton y Eisenberg [2000] proporcionan una guía de SQLJ, JDBC y tecnologías relacionadas. En http://www.sqlj.org se puede encontrar más información sobre SQLJ y software SQLJ. Date y Darwen [1997] y Date [1993a] incluyen una crítica de SQL-92.

Eisenberg y Melto [1999] proporcionan una visión general de SQL:1999. La norma está publicada como una secuencia de cinco documentos de la norma ISO/IEC, con otras partes que describen varias extensiones bajo desarrollo. La parte 1 (SQL/Framework) da una visión general de las otras partes. La parte 2 (SQL/Foundation) describe lo básico del lenguaje. La parte 3 (SQL/CLI) describe la interfaz en el nivel de llamada. La parte 4 (SQL/PSM) describe los módulos almacenados persistentes (Persistent Stored Modules, PSM), y la parte 5 (SQL/Bindings) describe los enlaces con el lenguaje anfitrión. La norma es útil para implementadores de bases de datos, pero es difícil de leer. Si se necesita, se puede comprar electrónicamente en el sitio Web http://webstore.ansi.org.

Muchos productos soportan las características de SQL además de las especificadas en las normas y muchos no soportan algunas características de la norma. Se puede encontrar más información sobre estas características en los manuales de usuario de SQL de los productos respectivos. http://java.sun.com/docs/books/tutorial es una fuente excelente para más información (actualizada) sobre JDBC y sobre Java en general. También hay disponibles en este URL referencias a libros sobre Java (incluyendo JDBC). La API ODBC se describe en Microsoft [1997] y Sanders [1998].

En los Capítulos 13 y 14 se estudia el procesamiento de consultas SQL, incluyendo algoritmos y estudios de rendimiento. También aparecen las notas bibliográficas relacionadas con este tema.

CAPÍTULO

5 OTROS LENGUAJES RELACIONALES

En el Capítulo 4 se ha descrito SQL, el lenguaje relacional de mayor influencia comercial. En este capítulo se estudiarán dos lenguajes más: QBE y Datalog. A diferencia de SQL, QBE es un lenguaje gráfico donde las consultas *parecen* tablas. QBE y sus variantes se usan ampliamente en sistemas de bases de datos para computadoras personales. Datalog tiene una sintaxis derivada del lenguaje Prolog. Aunque actualmente no se usa de forma comercial, Datalog se ha utilizado en el desarrollo de diversos sistemas de bases de datos.

En este capítulo se presentan las constructoras y conceptos fundamentales en lugar de un manual de usuario para estos lenguajes. Hay que tener presenta que las implementaciones individuales de un lenguaje puede diferir en los detalles, o puede dar soporte sólo a un subconjunto del lenguaje completo.

En este capítulo también se estudian interfaces de formularios y herramientas para generar informes y analizar datos. Aunque no son lenguajes estrictamente hablando, forman la interfaz principal a una base de datos para muchos usuarios. De hecho, la mayoría de usuarios en absoluto ejecutan consultas explícitas con un lenguaje de consulta y acceden a los datos mediante formularios, informes y otras herramientas de análisis de datos.

5.1. QUERY-BY-EXAMPLE

Query-by-Example (**QBE**, Consulta mediante ejemplos) es el nombre tanto de un lenguaje de manipulación de datos como el de un sistema de base de datos que incluyó a este lenguaje. El sistema de bases de datos QBE se desarrolló en el Centro de investigación T. J. Watson de IBM, a principios de los años setenta y el lenguaje de manipulación de datos QBE se usó más tarde en QMF (*Query Management Facility*, mecanismo de gestión de consultas), también de IBM. Actualmente, muchos de los sistemas de bases de datos para computadoras personales soportan variantes del lenguaje QBE. En este apartado se considera sólo el lenguaje de manipulación de datos. Tiene dos características distintivas:

1. A diferencia de muchos lenguajes de consulta y de programación, QBE presenta una **sintaxis bidimensional**. Las consultas *parecen* tablas. Una consulta en un lenguaje unidimensional (como SQL) se *puede* formular en una línea (posiblemente larga). Un lenguaje bidimensional *necesita* dos dimensiones para la formulación de consultas. (Existe una versión unidimensional de QBE, pero no se considerará en este estudio.)
2. Las consultas en QBE se expresan «mediante un ejemplo». En lugar de incluir un procedimiento para obtener la respuesta deseada, se usa un ejemplo de qué es lo deseado. El sistema generaliza este ejemplo para obtener la respuesta a la consulta.

A pesar de estas características tan poco comunes existe una correspondencia entre QBE y el cálculo relacional de dominios.

Las consultas en QBE se expresan utilizando **esqueletos de tablas**. Estos esqueletos de tablas presentan el esquema de la relación, como se muestra en la Figura 5.1. En lugar de llenar la pantalla con esqueletos de tablas, el usuario elige los esqueletos que necesita para una determinada consulta y rellena dichos esqueletos **con filas ejemplo**. Una fila ejemplo está formada por constantes y *elementos ejemplo*, que son variables de dominio. Para evitar confusiones, en QBE las variables de dominio van precedidas por un carácter de subrayado (_) como en *_x*, y las constantes aparecen sin ninguna indicación particular. Este convenio contrasta con la mayoría de los lenguajes, en los que las constantes se encierran entre comillas y las variables aparecen sin ninguna indicación.

5.1.1. Consultas sobre una relación

Recuperando el ejemplo bancario que se viene utilizando en capítulos anteriores, para obtener todos los números de préstamo de la sucursal Navacerrada se utilizará el esqueleto de la relación *préstamo* y se rellenará del modo siguiente:

préstamo	*número-préstamo*	*numbre-sucursal*	*importe*
	P._*x*	Navacerrada	