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 garantizar la fiabilidad de la información almacenada, a pesar de las caídas del sistema o de los intentos de acceso no autorizados. Si los datos van a ser compartidos entre diferentes 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 una gran cuerpo de conceptos y técnicas para la gestión de los datos. Estos conceptos y técnicas constituyen el objetivo central de este libro. 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 se usan ampliamente. Algunas de sus aplicaciones representativas son:

- *Banca*: para información de los clientes, cuentas, préstamos y transacciones bancarias.
- *Líneas aéreas*: para reservas e información de horarios. Las líneas aéreas fueron de las primeras en usar las bases de datos de forma distribuida geográficamente.
- *Universidades*: para información de los estudiantes, matrículas en las asignaturas y cursos.
- *Transacciones de tarjetas de crédito*: para compras con tarjeta de crédito y la generación de los extractos mensuales.
- *Telecomunicaciones*: para guardar un registro de las llamadas realizadas, generar las facturas mensuales, mantener el saldo de las tarjetas telefónicas de prepago y para almacenar información sobre las redes de comunicaciones.
- *Finanzas*: para almacenar información sobre compañías tenedoras, ventas y compras de productos financieros, como acciones y bonos; también para almacenar datos del mercado en tiempo real para permitir a los clientes la compraventa en línea y a la compañía la compraventa automática.
- *Ventas*: para información de clientes, productos y compras.

- *Comercio en línea*: para los datos de ventas ya mencionados y para el seguimiento de los pedidos Web, generación de listas de recomendaciones y mantenimiento de evaluaciones de productos en línea.
- *Producción*: para la gestión de la cadena de proveedores y para el seguimiento de la producción de artículos en las factorías, inventarios en los almacenes y pedidos.
- *Recursos humanos*: para información sobre los empleados, salarios, impuestos sobre los sueldos y prestaciones sociales, y para la generación de las nóminas.

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

Durante 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 interactuaban directamente con los sistemas de bases de datos, aunque sin darse cuenta interactuaban indirectamente con bases de datos—con informes impresos como los extractos de las tarjetas de crédito, o mediante agentes como los cajeros de los bancos y los agentes de reservas de las líneas aéreas. Después vinieron los cajeros automáticos y permitieron a los usuarios interactuar directamente con las bases de datos. Las interfaces telefónicas con las computadoras (sistemas de respuesta vocal interactiva) también permitieron a los usuarios tratar directamente con las bases de datos—la persona que llamaba podía marcar un número y pulsar las teclas del teléfono para introducir información o para seleccionar opciones alternativas, para conocer las horas de llegada o salida de los vuelos, por ejemplo, o para matricularse de asignaturas en una universidad.

La revolución de Internet a finales de los años noventa 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 dejaron disponibles en línea muchos servicios. Por ejemplo, cuando se accede a una librería en línea y se busca en una colección de libros o de música, se está accediendo a datos almacenados en una base de datos. Cuando se realiza un pedido en línea, el pedido se almacena en una base de datos. Cuando se accede al sitio Web de un banco y se consultan 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, puede que se recupere información personal de una base de datos para seleccionar los anuncios que se deben mostrar. Más aún, los datos sobre los accesos Web pueden almacenarse en una base de datos

Así, aunque las interfaces de usuario ocultan los 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 actualmente una parte esencial de la vida de casi todas las personas.

La importancia de los sistemas de bases de datos se puede juzgar de otra forma—actualmente, los fabricantes de sistemas de bases de datos como Oracle están entre las mayores compañías de software del 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 Propósito de los sistemas de bases de datos

Los sistemas de bases de datos surgieron en respuesta a los primeros métodos de gestión informatizada de los datos comerciales. A modo de ejemplo de dichos métodos, típicos de los años sesenta, considérese parte de una entidad bancaria que, entre otros datos, guarda información sobre todos los clientes y todas las cuentas de ahorro. Una manera de guardar la información en la computadora es almacenarla en archivos del sistema operativo. Para permitir que los usuarios manipulen la información, el sistema tiene varios programas de aplicación que gestionan los archivos, incluyendo programas para:

- Efectuar cargos o abonos en las cuentas.
- Añadir cuentas nuevas.
- Calcular el saldo de las cuentas.
- Generar los extractos mensuales.

Estos programas de aplicación los han escrito programadores de sistemas en respuesta a las necesidades del banco.

Se añaden nuevos programas de aplicación al sistema según surgen las necesidades. Por ejemplo, supóngase que una caja de ahorros decide ofrecer cuentas corrientes. En consecuencia, se crean nuevos archivos permanentes que contienen información acerca de todas las cuentas corrientes abiertas en el banco y puede que haya que escribir nuevos programas de aplicación para afrontar situaciones que no se dan en las cuentas de ahorro, como los descubiertos. Así, con el paso del tiempo, se añaden más archivos y programas de aplicación al sistema.

Los sistemas operativos convencionales soportan este **sistema de procesamiento de archivos** típico. El sistema almacena los registros permanentes en varios archivos y necesita diferentes programas de aplicación para extraer y añadir a los archivos correspondientes. Antes de la aparición de los sistemas gestores de bases de datos (SGBDs), las organizaciones normalmente almacenaban la información en sistemas de este tipo.

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

- **Redundancia e inconsistencia de los datos**. Debido a que los archivos y programas de aplicación los crean diferentes programadores en el transcurso de un largo período de tiempo, es probable que los diversos archivos tengan estructuras diferentes y que los programas estén escritos en varios lenguajes de programación diferentes. Además, puede que la información esté duplicada en varios lugares (archivos). Por ejemplo, la dirección y el número de teléfono de un cliente dado pueden aparecer en un archivo que contenga registros de cuentas de ahorros y en un archivo que contenga registros de cuentas corrientes. Esta redundancia conduce a costes de almacenamiento y de acceso más elevados. Además, puede dar lugar a la **inconsistencia de los datos**; es decir, puede que las diferentes copias de los mismos datos no coincidan. Por ejemplo, puede que el cambio en la dirección de un cliente esté reflejado en los registros de las cuentas de ahorro pero no 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 un código postal dado. El empleado pide al departamento de procesamiento de datos que genere esa lista. Debido a que esta petición no fue prevista por los diseñadores del sistema original, 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 extraer manualmente la información que necesita, o bien pedir a un programador de sistemas que escriba el programa de aplicación necesario. Ambas alternativas son obviamente insatisfactorias. Supóngase que se escribe el programa y que, varios días más tarde, el mismo empleado necesita reducir esa lista para que incluya únicamente a aquellos clientes que tengan una cuenta con saldo igual o superior a 10.000 €. Como se puede esperar, no existe ningún programa que genere tal lista. De nuevo, el empleado tiene que elegir entre dos opciones, ninguna de las cuales es satisfactoria.

  La cuestión aquí es que los entornos de procesamiento de archivos convencionales no permiten recuperar los datos necesarios de una forma práctica y eficiente. Hacen falta sistemas de recuperación de datos más adecuados para el uso general.

- **Aislamiento de datos.** Como 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 correspondientes.

- **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 ciertos tipos de cuentas bancarias no puede nunca ser inferior a una cantidad predeterminada (por ejemplo, 25 €). Los desarrolladores hacen cumplir esas restricciones en el sistema añadiendo el código correspondiente 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 se complica cuando las restricciones implican diferentes elementos de datos de diferentes archivos.

- **Problemas de atomicidad.** Los sistemas informáticos, como cualquier otro dispositivo mecánico o eléctrico, está sujeto a fallos. En muchas aplicaciones es crucial asegurar que, si se produce algún fallo, los datos se restauren al estado consistente que existía antes del fallo. Considérese un programa para transferir 50 € desde la cuenta *A* a la *B*. Si se produce un fallo del sistema durante la ejecución del programa, es posible que los 50 € fueran retirados de la cuenta *A* pero no abonados en la cuenta *B*, dando lugar a un estado inconsistente de la base de datos. Evidentemente, resulta esencial para la consistencia de la base de datos que tengan lugar tanto el abono como el cargo, o que no tenga lugar ninguno. Es decir, la transferencia de fondos debe ser *atómica*—debe ocurrir en su totalidad o no ocurrir en absoluto. Resulta difícil asegurar la atomicidad en los sistemas convencionales de procesamiento de archivos.

- **Anomalías en el acceso concurrente**. Para aumentar el rendimiento global del sistema y obtener una respuesta más rápida, muchos sistemas permiten que varios usuarios actualicen los datos simultáneamente. En realidad, hoy en día, los principales sitios de comercio electrónico de Internet pueden tener millones de accesos diarios de compradores a sus datos. En tales entornos es posible la interacción de actualizaciones concurrentes y puede dar lugar a datos inconsistentes. Considérese una cuenta bancaria *A*, que contenga 500 €. Si dos clientes retiran fondos (por ejemplo, 50 € y 100 €, respectivamente) de la cuenta *A* aproximadamente al mismo tiempo, el resultado de las ejecuciones concurrentes puede dejar la cuenta en un estado incorrecto (o inconsistente). Supóngase que los programas que se ejecutan para cada retirada leen el saldo anterior, reducen su valor en el importe que se retira y luego escriben el resultado. Si los dos programas se ejecutan concurrentemente, pueden leer el valor 500 €, y escribir después 450 € y 400 €, respectivamente. Dependiendo de cuál escriba el valor en último lugar, la cuenta puede contener 450 € o 400 €, en lugar del valor correcto, 350 €. Para protegerse contra esta posibilidad, el sistema debe mantener alguna forma de supervisión. Pero es difícil ofrecer supervisión, ya que muchos programas de aplicación diferentes que no se han coordinado con anterioridad pueden tener acceso a los datos.

- **Problemas de seguridad**. No todos los usuarios de un sistema de bases de datos deben poder acceder a todos los datos. Por ejemplo, en un sistema bancario, el personal de nóminas sólo necesita ver la parte de la base de datos que contiene información acerca de los diferentes empleados del banco. No necesitan tener acceso a la información acerca de las cuentas de clientes. Pero, como los programas de aplicación se añaden al sistema de procesamiento de datos de una forma ad hoc, es difícil hacer cumplir tales restricciones de seguridad.

Estas dificultades, entre otras, motivaron el desarrollo de los sistemas de bases de datos. En el resto del libro se examinarán los conceptos y los algoritmos que permiten que los sistemas de bases de datos resuelvan los problemas de los sistemas de procesamiento de archivos. En general, en este libro se usa una entidad bancaria como ejemplo de aplicación típica de procesamiento de datos que puede encontrarse en una empresa.

1.3 Visión de los datos

Un sistema de bases de datos es una colección de datos interrelacionados y un conjunto de programas que permiten a los usuarios tener acceso a esos datos y modificarlos. Una de las principales finalidades de los sistemas de bases de datos es ofrecer a los usuarios una visión *abstracta* de los datos. Es decir, el sistema oculta ciertos detalles del modo en que se almacenan y mantienen los datos.

1.3.1 Abstracción de datos

Para que el sistema sea útil debe recuperar los datos eficientemente. La necesidad de eficiencia ha llevado a los diseñadores a usar estructuras de datos complejas para la representación de los datos en la base de datos. Dado que muchos de los usuarios de sistemas de bases de datos no tienen formación en informática, los desarrolladores ocultan esa complejidad a los usuarios mediante varios niveles de abstracción para simplificar la interacción de los usuarios con el sistema:

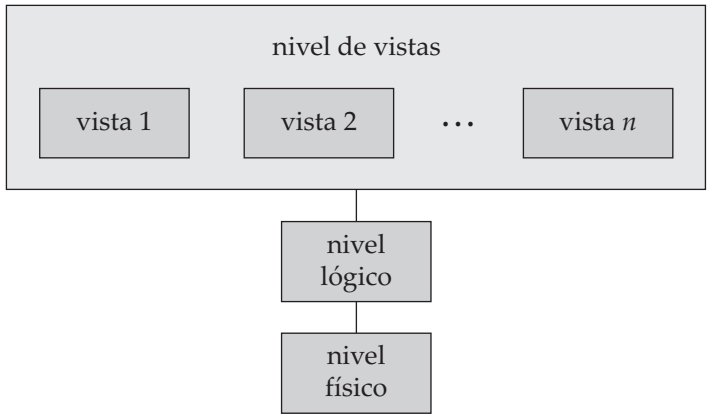


Figura 1.1 Los tres niveles de abstracción de datos.

- **Nivel físico**. El nivel más bajo de abstracción describe *cómo* se almacenan realmente los datos. El nivel físico describe en detalle las estructuras de datos complejas de bajo nivel.
- **Nivel lógico**. El nivel inmediatamente superior de abstracción describe *qué* datos se almacenan en la base de datos y qué relaciones existen entre esos datos. El nivel lógico, por tanto, describe toda la base de datos en términos de un número pequeño de estructuras relativamente simples. Aunque la implementación de esas 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 guarda en la base de datos, usan el nivel de abstracción lógico.
- **Nivel de vistas**. El nivel más elevado de abstracción sólo describe parte de la base de datos. Aunque el nivel lógico usa estructuras más simples, queda algo de complejidad debido a la variedad de información almacenada en las grandes bases de datos. Muchos usuarios del sistema de bases de datos no necesitan toda esta información; en su lugar sólo necesitan tener acceso a una parte de la base de datos. El nivel de abstracción de vistas existe para simplificar su interacción con el sistema. 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 diferencia entre los niveles de abstracción. La mayoría de los lenguajes de programación de alto nivel soportan el concepto de tipo estructurado. Por ejemplo, en los lenguajes tipo Pascal se pueden declarar registros de la manera siguiente:

```
type cliente = record
                 id_cliente : string;
                 nombre_cliente : string;
                 calle_cliente : string;
                 ciudad_cliente : string;
               end;
```

Este código define un nuevo tipo de registro denominado *cliente* con cuatro campos. Cada campo tiene un nombre y un tipo asociados. Una entidad bancaria puede tener varios tipos de estos registros, incluidos:

- *cuenta*, con los campos *número_cuenta* y *saldo*.
- *empleado*, con los campos *nombre_empleado* y *sueldo*.

En el nivel físico, los registros *cliente*, *cuenta* o *empleado* se pueden describir como bloques de posiciones consecutivas de almacenamiento (por ejemplo, palabras o bytes). El compilador oculta este nivel

de detalle a los programadores. De manera parecida, el sistema de base de datos oculta muchos de los detalles de almacenamiento de los niveles inferiores a los programadores de bases de datos. Los administradores de bases de datos, por otro lado, 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 en el fragmento de código anterior, y también se define la relación entre estos tipos de registros. Los programadores que usan un lenguaje de programación trabajan en este nivel de abstracción. De manera parecida, los administradores de bases de datos suelen trabajar en este nivel de abstracción.

Finalmente, en el nivel de vistas, los usuarios de computadoras ven un conjunto de programas de aplicación que ocultan los detalles de los tipos de datos. De manera parecida, en el nivel de vistas se definen varias vistas de la base de datos y los usuarios de la base de datos pueden verlas. Además de ocultar los detalles del nivel lógico de la base de datos, las vistas también proporcionan un mecanismo de seguridad para evitar que los usuarios tengan acceso a ciertas partes de la base de datos. Por ejemplo, los cajeros de un banco sólo ven la parte de la base de datos que contiene información de las cuentas de los clientes; no pueden tener acceso 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 se elimina. La colección de información almacenada en la base de datos en un momento dado se denomina **ejemplar** de la base de datos. El diseño general de la base de datos se denomina **esquema** de la base de datos. Los esquemas se modifican rara vez, si es que se modifican.

El concepto de esquemas y ejemplares de las bases de datos se puede comprender por analogía con los programas escritos en un lenguaje de programación. El esquema de la base de datos se corresponde con las declaraciones de las variables (junto con las definiciones de tipos asociadas) de los programas. Cada variable tiene un valor concreto en un instante dado. Los valores de las variables de un programa en un instante dado se corresponden con un *ejemplar* del esquema de la base de datos.

Los sistemas de bases de datos tiene varios esquemas divididos según los niveles de abstracción. El **esquema físico** describe el diseño de la base de datos en el nivel físico, mientras que el **esquema lógico** describe su diseño en el nivel lógico. Las bases de datos también pueden tener varios esquemas en el nivel de vistas, a veces 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 sobre los programas de aplicación, ya que los programadores crean las aplicaciones usando el esquema lógico. El esquema físico está oculto bajo el esquema lógico, y generalmente puede modificarse fácilmente sin afectar a los programas de aplicación. Se dice que los programas de aplicación muestran **independencia!física respecto de los datos** si no dependen del esquema físico y, por tanto, no hace falta volver a escribirlos si se modifica el esquema físico.

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

1.3.3 Modelos de datos

Bajo la estructura de las bases de datos se encuentra el **modelo de datos**: una colección de herramientas conceptuales para describir los datos, sus relaciones, su semántica y las restricciones de consistencia. Los modelos de datos ofrecen un modo de describir el diseño de las bases de datos en los niveles físico, lógico y de vistas.

En este texto se van a tratar varios modelos de datos diferentes. Los modelos de datos pueden clasificarse en cuatro categorías diferentes:

- **Modelo relacional**. El modelo relacional usa una colección de tablas para representar tanto los datos como sus relaciones. Cada tabla tiene varias columnas, y cada columna tiene un nombre único. 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 dado. Cada tipo de registro define un número fijo de campos, o atributos. Las columnas de la tabla se corresponden con los atributos

del tipo de registro. El modelo de datos relacional es el modelo de datos más ampliamente usado, y una gran mayoría de sistemas de bases de datos actuales se basan en el modelo relacional. Los Capítulos 2 al 7 tratan el modelo relacional en detalle.

- **El modelo entidad-relación**. El modelo de datos entidad-relación (E-R) se basa en una percepción del mundo real que consiste en una colección de objetos básicos, denominados *entidades*, y de las *relaciones* entre ellos. Una entidad es una "cosa" u "objeto" del mundo real que es distinguible de otros objetos. El modelo entidad-relación se usa mucho en el diseño de bases de datos y en el Capítulo 6 se examina detalladamente.
- **Modelo de datos orientado a objetos**. 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 considerar como una extensión del modelo E-R con los conceptos de la encapsulación, los métodos (funciones) y la identidad de los objetos. En el Capítulo 9 se examina este modelo de datos.
- **Modelo de datos semiestructurados**. El modelo de datos semiestructurados permite la especificación de datos donde los elementos de datos individuales del mismo tipo pueden tener diferentes conjuntos de atributos. Esto lo diferencia 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 emplea mucho para representar datos semiestructurados. Se estudia en el Capítulo 10.

El **modelo de datos de red** y el **modelo de datos jerárquico** precedieron cronológicamente al relacional. Estos modelos estuvieron íntimamente ligados a la implementación subyacente y complicaban la tarea del modelado de datos. En consecuencia, se usan muy poco hoy en día, excepto en el código de bases de datos antiguas que sigue estando en servicio en algunos lugares. Se describen brevemente en los Apéndices A y B para los lectores interesados.

1.4 Lenguajes de bases de datos

Los sistemas de bases de datos proporcionan 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 y las modificaciones de la base de datos. En la práctica, los lenguajes de definición y manipulación de datos no son dos lenguajes diferentes; en cambio, simplemente forman parte de un único lenguaje de bases de datos, como puede ser el muy usado SQL.

1.4.1 Lenguaje de manipulación de datos

Un **lenguaje de manipulación de datos** (LMD) es un lenguaje que permite a los usuarios tener acceso a los datos organizados mediante el modelo de datos correspondiente o manipularlos. Los tipos de acceso son:

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

Hay fundamentalmente dos tipos:

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

Los LMDs declarativos suelen resultar más fáciles de aprender y de usar que los procedimentales. Sin embargo, como el usuario no tiene que especificar cómo conseguir los datos, el sistema de bases de datos tiene que determinar un medio eficiente de acceso a los datos.

Una **consulta** es una instrucción que solicita que se recupere información. La parte de los LMDs implicada en la recuperación de información se denomina **lenguaje de consultas**. Aunque técnicamente sea incorrecto, resulta habitual usar las expresiones *lenguaje de consultas* y *lenguaje de manipulación de datos* como sinónimas.

Existen varios lenguajes de consultas de bases de datos en uso, tanto comercial como experimentalmente. El lenguaje de consultas más ampliamente usado, SQL, se estudiará en los Capítulos 3 y 4. También se estudiarán otros lenguajes de consultas en el Capítulo 5.

Los niveles de abstracción que se trataron en el Apartado 1.3 no sólo se aplican a la definición o estructuración de datos, sino también a su manipulación. En el nivel físico se deben definir los algoritmos que permitan un acceso eficiente a los datos. En los niveles superiores de abstracción se pone el énfasis en la facilidad de uso. El objetivo es permitir que los seres humanos interactúen de manera 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.

1.4.2 Lenguaje de definición de datos

Los esquemas de las bases de datos se especifican mediante un conjunto de definiciones expresadas mediante un lenguaje especial denominado **lenguaje de definición de datos (LDD)**. El LDD también se usa para especificar más propiedades de los datos.

La estructura de almacenamiento y los métodos de acceso usados por el sistema de bases de datos se especifican mediante 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 las bases de datos, que suelen ocultarse a los usuarios.

Los valores de los 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 las comprueban cada vez que se modifica la base de datos. En general, las restricciones pueden ser predicados arbitrarios relativos a la base de datos. No obstante, los predicados arbitrarios pueden resultar costosos de comprobar. Por tanto, los sistemas de bases de datos se concentran en las restricciones de integridad que pueden comprobarse con una sobrecarga mínima:

- **Restricciones de dominio.** Se debe asociar un dominio de valores posibles a cada atributo (por ejemplo, tipos enteros, tipos de carácter, tipos fecha/hora). La declaración de un atributo como parte de un dominio concreto actúa como restricción de los valores que puede adoptar. Las restricciones de dominio son la forma más elemental de restricción de integridad. El sistema las comprueba fácilmente siempre que se introduce un nuevo elemento de datos en la base de datos.
- **Integridad referencial.** Hay casos en los que se desea asegurar que un valor que aparece en una relación para un conjunto de atributos dado aparece también para un determinado conjunto de atributos en otra relación (integridad referencial). Las modificaciones de la base de datos pueden causar violaciones de la integridad referencial. Cuando se viola una restricción de integridad, el procedimiento normal es rechazar la acción que ha causado esa violación.
- **Asertos.** Un aserto es cualquier condición que la base de datos debe satisfacer siempre. Las restricciones de dominio y las restricciones de integridad referencial son formas especiales de asertos. No obstante, hay muchas restricciones que no pueden expresarse empleando únicamente esas formas especiales. Por ejemplo: "Cada préstamo tiene como mínimo un cliente tenedor de una cuenta con un saldo mínimo de 1.000,00 €" debe expresarse en forma de aserto. Cuando se crea un aserto, el sistema comprueba su validez. Si el aserto es válido, cualquier modificación futura de la base de datos se permite únicamente si no hace que se viole ese aserto.
- **Autorización.** Puede que se desee diferenciar entre los usuarios en cuanto al tipo de acceso que se les permite a diferentes valores de los datos de la base de datos. Estas diferenciaciones se

expresan en términos de **autorización**, cuyas modalidades más frecuentes son: **autorización de lectura**, que permite la lectura pero no la modificación de los datos; **autorización de inserción**, que permite la inserción de datos nuevos, pero no la modificación de los datos ya existentes; **autorización de actualización**, que permite la modificación, pero no la eliminación, de los datos; y la **autorización de eliminación**, que permite la eliminación de datos. A cada usuario se le pueden asignar todos, ninguno o una combinación de estos tipos de autorización.

El LDD, al igual que cualquier otro lenguaje de programación, obtiene como entrada algunas instrucciones y genera una salida. La salida del LDD se coloca en el **diccionario de datos**, que contiene **metadatos**—es decir, datos sobre datos. El diccionario de datos se considera un tipo especial de tabla, a la que sólo puede tener acceso y actualizar el propio sistema de bases de datos (no los usuarios normales). El sistema de bases de datos consulta el diccionario de datos antes de leer o modificar los datos reales.

1.5 Bases de datos relacionales

Las bases de datos relacionales se basan en el modelo relacional y usan un conjunto de tablas para representar tanto los datos como las relaciones entre ellos. También incluyen un LMD y un LDD. La mayor parte de los sistemas de bases de datos relacionales comerciales emplean el lenguaje SQL, que se trata en este apartado y que se tratará con gran detalle en los Capítulos 3 y 4. En el Capítulo 5 se estudiarán otros lenguajes influyentes.

1.5.1 Tablas

Cada tabla tiene varias columnas, y cada columna tiene un nombre único. En la Figura 1.2 se presenta un ejemplo de base de datos relacional consistente en tres tablas: una muestra detalles de los clientes de un banco, la segunda muestra las cuentas y la tercera muestra las cuentas que pertenecen a cada cliente.

La primera tabla, la tabla *cliente*, muestra, por ejemplo, que el cliente identificado por *id_cliente* 19.283.746 se llama González y vive en la calle Arenal en La Granja. La segunda tabla, *cuenta*, muestra, por ejemplo, que la cuenta C-101 tiene un saldo de 500 € y la C-201 un saldo de 900 €.

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

El modelo relacional es un ejemplo de 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 dado. Cada tipo de registro define un número fijo de campos, o atributos. Las columnas de la tabla se corresponden con los atributos del tipo de registro.

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

El modelo de datos relacional es el modelo de datos más ampliamente usado, y una gran mayoría de los sistemas de bases de datos actuales se basan en el modelo relacional. Los Capítulos 2 al 7 tratan el modelo relacional con detalle.

Obsérvese también que en el modelo relacional es posible crear esquemas que tengan problemas tales como información duplicada innecesariamente. Por ejemplo, supóngase que se almacena *número_cuenta* como atributo de un registro *cliente*. Entonces, para representar el hecho de que tanto la cuentas C-101 como la cuenta C-201 pertenece al cliente González (con *id_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 el modo de distinguir los buenos diseños de esquema de los malos.

id_cliente	*nombre_cliente*	*calle_cliente*	*ciudad_cliente*
19.283.746	González	Arenal, 12	La Granja
67.789.901	López	Mayor, 3	Peguerinos
18.273.609	Abril	Preciados, 123	Valsaín
32.112.312	Santos	Mayor, 100	Peguerinos
33.666.999	Rupérez	Ramblas, 175	León
01.928.374	Gómez	Carretas, 72	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

(c) La tabla *impositor*

Figura 1.2 Un ejemplo de base de datos relacional.

1.5.2 Lenguaje de manipulación de datos

El lenguaje de consultas de SQL no es procedimental. Usa como entrada varias tablas (posiblemente sólo una) y devuelve siempre una sola tabla. A continuación se ofrece un ejemplo de consulta SQL que halla el nombre de todos los clientes que residen en Peguerinos:

select *cliente.nombre_cliente*
from *cliente*
where *cliente.ciudad_cliente* = 'Peguerinos'

La consulta especifica que hay que recuperar (*select*) las filas de (*from*) la tabla *cliente* en las que (*where*) la *ciudad_cliente* es Peguerinos, y que sólo debe mostrarse el atributo *nombre_cliente* de esas filas. Más concretamente, el resultado de la ejecución de esta consulta es una tabla con una sola columna denominada *nombre_cliente* y un conjunto de filas, cada una de las cuales contiene el nombre de un cliente cuya *ciudad_cliente* es Peguerinos. Si la consulta se ejecuta sobre la tabla de la Figura 1.2, el resultado constará de dos filas, una con el nombre López y otra con el nombre Santos.

Las consultas pueden involucrar información de más de una tabla. Por ejemplo, la siguiente consulta busca todos los números de cuenta y sus saldos del cliente con *id_cliente* 19.283.746.

```
select cuenta.número_cuenta, 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 sobre las tablas de la Figura 1.2, el sistema encontraría que las dos cuentas denominadas C-101 y C-201 pertenecen al cliente 19.283.746 y el resultado consistiría en una tabla con dos columnas (*número_cuenta, saldo*) y dos filas (C-101, 500) y (C-201,900).

1.5.3 Lenguaje de definición de datos

SQL ofrece un LDD elaborado que permite definir tablas, restricciones de integridad, asertos, etc.

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

```
create table cuenta
      (número_cuenta char(10),
       saldo integer)
```

La ejecución de esta instrucción LDD crea la tabla *cuenta.* Además, actualiza el diccionario de datos, que contiene metadatos (1.4.2). Los esquemas de las tablas son ejemplos de metadatos.

1.5.4 Acceso a las bases de datos desde los programas de aplicación

SQL no es tan potente como la máquina universal de Turing; es decir, hay algunos cálculos que no pueden obtenerse mediante ninguna consulta SQL. Esos cálculos deben escribirse en un lenguaje *anfitrión*, como Cobol, C, C++ o Java. Los **programas de aplicación** son programas que se usan para interactuar de esta manera con las bases de datos. Algunos de los ejemplos de un sistema bancario serían los programas que generan las nóminas, realizan cargos en las cuentas, realizan abonos en las cuentas o transfieren fondos entre las cuentas.

Para tener acceso a la base de datos, las instrucciones LMD deben ejecutarse desde el lenguaje anfitrión. Hay dos maneras de conseguirlo:

- Proporcionando una interfaz de programas de aplicación (conjunto de procedimientos) que se pueda 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 su empleo con el lenguaje C es un estándar de interfaz de programas de aplicación usado habitualmente. El estándar de conectividad de Java con bases de datos (JDBC, Java Data Base Connectivity) ofrece las características correspondientes para el lenguaje Java.

- Extendiendo la sintaxis del lenguaje anfitrión para que incorpore las llamadas LMD dentro del programa del lenguaje anfitrión. Generalmente, un carácter especial precede a las llamadas LMD y un preprocesador, denominado **precompilador LMD**, convierte las instrucciones LMD en llamadas normales a procedimientos en el lenguaje anfitrión.

1.6 Diseño de bases de datos

Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de información. Esas grandes cantidades de información no existen aisladas. Forman parte del funcionamiento de alguna empresa, cuyo producto final puede que sea la información obtenida de la base de datos o algún dispositivo o servicio para el que la base de datos sólo desempeña un papel secundario.

El diseño de bases de datos implica principalmente el diseño del esquema de las bases de datos. El diseño de un entorno completo de aplicaciones para la base de datos que satisfaga las necesidades de la empresa que se está modelando exige prestar atención a un conjunto de aspectos más amplio. Este texto se centrará inicialmente en la escritura de las consultas a la base de datos y en el diseño de los esquemas de las bases de datos. En el Capítulo 8 se estudia el proceso general de diseño de las aplicaciones.

1.6.1 Proceso de diseño

Los modelos de datos de alto nivel resultan útiles a los diseñadores de bases de datos al ofrecerles un marco conceptual en el que especificar, de manera sistemática, los requisitos de datos de los usuarios de las bases de datos y la manera en que se estructurará la base de datos para satisfacer esos requisitos. La fase inicial del diseño de las bases de datos, por tanto, es caracterizar completamente los requisitos de datos de los hipotéticos usuarios de la base de datos. Los diseñadores de bases de datos deben interactuar ampliamente con los expertos y usuarios del dominio para llevar a cabo esta tarea. El resultado de esta fase es la especificación de los requisitos de los usuarios.

A continuación, el diseñador escoge un modelo de datos y, mediante la aplicación de los conceptos del modelo de datos elegido, traduce esos requisitos en un esquema conceptual de la base de datos. El esquema desarrollado en esta fase de **diseño conceptual** ofrece una visión general detallada de la empresa. El diseñador revisa el esquema para confirmar que todos los requisitos de datos se satisfacen realmente y no entran en conflicto entre sí. El diseñador también puede examinar el diseño para eliminar cualquier característica redundante. En este punto, la atención se centra en describir los datos y sus relaciones, más que en especificar los detalles del almacenamiento físico.

En términos del modelo relacional, el proceso de diseño conceptual implica decisiones sobre *qué* atributos se desea capturar en la base de datos y *cómo agruparlos* para formar las diferentes tablas. La parte "qué" es, esencialmente, una decisión conceptual, y no se seguirá estudiando en este texto. La parte del "cómo" es, esencialmente, un problema informático. Hay dos vías principales para afrontar el problema. La primera supone usar el modelo entidad-relación (Apartado 1.6.3); la otra es emplear un conjunto de algoritmos (denominados colectivamente como *normalización*) que toma como entrada el conjunto de todos los atributos y genera un conjunto de tablas (Apartado 1.6.4).

Un esquema conceptual completamente desarrollado también indica los requisitos funcionales de la empresa. En la **especificación de requisitos funcionales** los usuarios describen el tipo de operaciones (o transacciones) que se llevarán a cabo con los datos. Un ejemplo de estas operaciones es modificar o actualizar los datos, buscar y recuperar datos concretos y eliminar datos. En esta etapa del diseño conceptual el diseñador puede revisar el esquema para asegurarse de que satisface los requisitos funcionales.

El proceso de pasar de un modelo de datos abstracto a la implementación de la base de datos continúa con dos fases de diseño finales. En la **fase de diseño lógico** el diseñador relaciona el esquema conceptual de alto nivel con el modelo de implementación de datos del sistema de bases de datos que se va a usar. El diseñador usa el esquema de bases de datos específico para el sistema resultante en la **fase de diseño físico** posterior, en la que se especifican las características físicas de la base de datos. Entre esas características están la forma de organización de los archivos y las estructuras de almacenamiento interno; se estudian en el Capítulo 11.

1.6.2 Diseño de la base de datos para una entidad bancaria

Para ilustrar el proceso de diseño, examinemos el modo en que puede diseñarse una base de datos para una entidad bancaria. La especificación inicial de los requisitos de los usuarios puede basarse en entrevistas con los usuarios de la base de datos y en el análisis de la entidad realizado por el propio diseñador. La descripción que surge de esta fase de diseño sirve como base para especificar la estructura conceptual de la base de datos. Éstas son las principales características de la entidad bancaria:

- El banco está organizado en sucursales. Cada sucursal se ubica en una ciudad determinada y queda identificada por un nombre único. El banco supervisa los activos de cada sucursal.
- Los clientes quedan identificados por el valor de su *id_cliente.* El banco almacena el nombre de cada cliente y la calle y la ciudad en la que vive. Los clientes pueden abrir cuentas y solicitar préstamos. Cada cliente puede asociarse con un empleado del banco en concreto, que puede actuar como prestamista o como asesor personal para ese cliente.
- El banco ofrece dos tipos de cuenta: de ahorro y corriente. Las cuentas pueden tener como titular a más de un cliente, y cada cliente puede abrir más de una cuenta. A cada cuenta se le asigna un número de cuenta único. El banco guarda un registro del saldo de cada cuenta y de la fecha

más reciente en que cada cliente titular de esa cuenta tuvo acceso a ella. Además, cada cuenta de ahorro tiene una tasa de interés y se registran los descubiertos de las cuentas corrientes.

- El banco ofrece préstamos a sus clientes. Cada préstamo se origina en una sucursal concreta y puede tener como titulares a uno o más clientes. Cada préstamo queda identificado por un número de préstamo único. De cada préstamo el banco realiza un seguimiento del importe del préstamo y de sus pagos. Aunque el número de pago del préstamo no identifica de manera única un pago concreto entre todos los del banco, el número de pago identifica cada pago concreto de un préstamo dado. Se registran la fecha y el importe de cada pago.
- Los empleados del banco quedan identificados por el valor de su *id_empleado*. La administración del banco almacena el nombre y número de teléfono de cada empleado, el nombre de las personas dependientes de cada empleado y el número de *id_empleado* del jefe de cada empleado. El banco también realiza un seguimiento de la fecha de contratación y, por tanto, de la antigüedad de cada empleado.

En las entidades bancarias reales, el banco realizaría un seguimiento de los depósitos y de los reintegros de las cuentas de ahorro y corrientes, igual que realiza un seguimiento de los pagos de las cuentas de crédito. Dado que los requisitos de modelado para ese seguimiento son parecidas y nos gustaría que la aplicación de ejemplo no tuviera un tamaño excesivo, en nuestro modelo no se realiza el seguimiento de esos depósitos y reintegros.

1.6.3 El modelo entidad-relación

El modelo de datos entidad-relación (E-R) está basado en una percepción del mundo real que consiste en un conjunto de objetos básicos, denominados *entidades*, y de las *relaciones* entre esos objetos. Una entidad es una "cosa" u "objeto" del mundo real que es distinguible de otros objetos. Por ejemplo, cada persona es una entidad, y las cuentas bancarias pueden considerarse entidades.

Las entidades se describen en las bases de datos mediante un conjunto de **atributos**. Por ejemplo, los atributos *número_cuenta* y *saldo* pueden describir una cuenta concreta de un banco y constituyen atributos del conjunto de entidades *cuenta*. Análogamente, los atributos *nombre_cliente*, *calle_cliente* y *ciudad_cliente* pueden describir una entidad *cliente*.

Se usa un atributo extra, *id_cliente*, para identificar unívocamente a los clientes (dado que es posible que haya dos clientes con el mismo nombre, calle y ciudad). Se debe asignar un identificador de cliente único a cada cliente. En Estados Unidos, muchas empresas usan el número de la seguridad social de cada persona (un número único que el Gobierno de Estados Unidos asigna a cada persona) como identificador de cliente.

Una **relación** es una asociación entre varias entidades. Por ejemplo, la 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 (esquema) de la base de datos se puede expresar gráficamente mediante un *diagrama E-R*, que está constituido por los siguientes componentes:

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

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

Como ilustración, considérese parte de un sistema bancario de bases de datos consistente en los clientes y las cuentas que tienen esos clientes. La Figura 1.3 muestra el diagrama E-R correspondiente. El

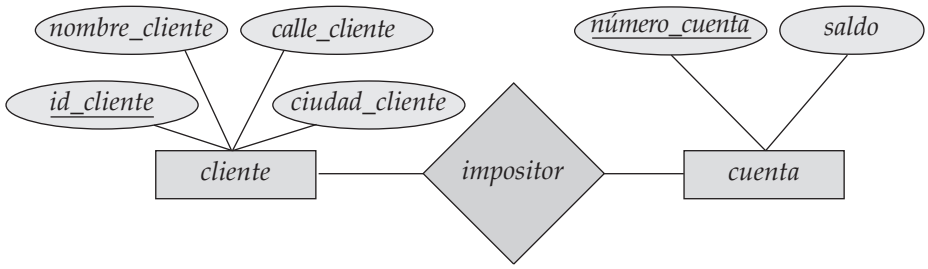


Figura 1.3 Un ejemplo de diagrama E-R.

id_cliente	*número_cuenta*	*saldo*
19.283.746	C-101	500
19.283.746	C-201	900
01.928.374	C-215	700
67.789.901	C-102	400
18.273.609	C-305	350
32.112.312	C-217	750
33.666.999	C-222	700
01.928.374	C-201	900

Figura 1.4 La tabla *impositor'*.

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 sólo debe pertenecer a un cliente, el modelo puede expresar esa restricción.

El modelo entidad-relación se usa ampliamente en el diseño de bases de datos, y en el Capítulo 6 se explora en detalle.

1.6.4 Normalización

Otro método de diseño de bases de datos es usar un proceso que suele denominarse normalización. El objetivo es generar un conjunto de esquemas de relaciones que permita almacenar información sin redundancias innecesarias, pero que también permita recuperar la información con facilidad. El enfoque es diseñar esquemas que se hallen en la *forma normal* adecuada. Para determinar si un esquema de relación se halla en una de las formas normales deseadas, hace falta información adicional sobre la empresa real que se está modelando con la base de datos. El enfoque más frecuente es usar **dependencias funcionales**, que se tratan en el Apartado 7.4.

Para comprender la necesidad de la normalización, obsérvese lo que puede fallar en un mal diseño de base de datos. Algunas de las propiedades no deseables de un mal son:

- Repetición de la información.
- Imposibilidad de representar determinada información.

Se examinarán estos problemas con la ayuda de un diseño de bases de datos modificado para el ejemplo bancario.

Supóngase que, en lugar de tener las dos tablas separadas *cuenta* e *impositor*, se tiene una sola tabla, *impositor'*, que combina la información de las dos tablas (como puede verse en la Figura 1.4). Obsérvese que hay dos filas de *impositor'* que contienen información sobre la cuenta C-201. La repetición de información en este diseño alternativo no es deseable. La repetición de información malgasta espacio.

id_cliente	*nombre_cliente*	*calle_cliente*	*ciudad_cliente*	*número_cuenta*
19.283.746	González	Arenal, 12	La Granja	C-101
19.283.746	González	Arenal, 12	La Granja	C-201
67.789.901	López	Mayor, 3	Peguerinos	C-102
18.273.609	Abril	Preciados, 123	Valsaín	C-305
32.112.312	Santos	Mayor, 100	Peguerinos	C-217
33.666.999	Rupérez	Ramblas, 175	León	C-222
01.928.374	Gómez	Carretas, 72	Cerceda	C-201

Figura 1.5 La tabla *cliente'*.

Además, complica las actualizaciones de la base de datos. Supóngase que se desea modificar el saldo de la cuenta C-201 de 900 a 950 €. Esta modificación debe reflejarse en las dos filas; compárese con el diseño original, en el que esto daría lugar a la actualización de una sola fila. Por tanto, las actualizaciones resultan más costosas bajo el diseño alternativo que bajo el diseño original. Cuando se lleva a cabo la actualización de la base de datos alternativa, hay que asegurarse de que *todas* las tuplas que afectan a la cuenta C-201 se actualicen, o la base de datos mostrará dos valores de saldo diferentes para la cuenta C-201.

Examínese ahora el problema de la "imposibilidad de representar determinada información". Supóngase que, en vez de tener las dos tablas separadas *cliente* e *impositor*, se tuviera una sola tabla, *cliente'*, que combinara la información de esas dos tablas (como puede verse en la Figura 1.5). No se puede representar directamente la información relativa a los clientes (*id_cliente*, *nombre_cliente*, *calle_cliente*, *ciudad_cliente*) a menos que el cliente tenga, como mínimo, una cuenta en el banco. Esto se debe a que las filas de *cliente'* necesitan valores de *número_cuenta*.

Una solución de este problema es introducir valores **nulos**. Los valores *nulos* indican que el valor no existe (o es desconocido). Los valores desconocidos pueden ser valores *ausentes* (el valor existe, pero no se tiene la información) o valores *desconocidos* (no se sabe si el valor existe realmente o no). Como se verá más adelante, los valores nulos resultan difíciles de tratar, y es preferible no recurrir a ellos. Si no se desea tratar con valores nulos, se puede crear un elemento concreto de información del cliente sólo si el cliente tiene cuenta en el banco (obsérvese que los clientes pueden tener un préstamo pero no tener ninguna cuenta). Además, habría que eliminar esa información cuando el cliente cerrara la cuenta. Claramente, esta situación no es deseable ya que, bajo el diseño original de la base de datos, la información de los clientes estaría disponible independientemente de si el cliente tiene cuenta en el banco o no, y sin necesidad de recurrir a los valores nulos.

1.7 Bases de datos basadas en objetos y semiestructuradas

Varias áreas de aplicaciones de los sistemas de bases de datos están limitadas por las restricciones del modelo de datos relacional. En consecuencia, los investigadores han desarrollado varios modelos de datos para tratar con estos dominios de aplicación. Los modelos de datos que se tratarán en este texto son el orientado a objetos y el relacional orientado a objetos, representativos de los modelos de datos basados en objetos, y XML, representativo de los modelos de datos semiestructurados.

1.7.1 Modelos de datos basados en objetos

El **modelo de datos orientado a objetos** se basa en el paradigma de los lenguajes de programación orientados a objetos, que actualmente se usa en gran medida. La herencia, la identidad de los objetos y la encapsulación (ocultación de la información), con métodos para ofrecer una interfaz para los objetos, están entre los conceptos principales de la programación orientada a objetos que han encontrado aplicación en el modelado de datos. El modelo de datos orientado a objetos también soporta un sistema elaborado de tipos, incluidos los tipos estructurados y las colecciones. El modelo orientado a objetos puede considerarse una extensión del modelo E-R con los conceptos de encapsulación, métodos (funciones) e identidad de los objetos.

El **modelo de datos relacional orientado a objetos** extiende el modelo relacional tradicional con gran variedad de características como los tipos estructurados y las colecciones, así como la orientación a objetos.

En el Capítulo 9 se examinan las bases de datos relacionales orientadas a objetos (es decir, las bases de datos construidas según el modelo relacional orientado a objetos), así como las bases de datos orientadas a objetos (es decir, las bases de datos construidas según el modelo de datos orientado a objetos).

1.7.2 Modelos de datos semiestructurados

Los modelos de datos semiestructurados permiten la especificación de los datos en los que cada elemento de datos del mismo tipo puede tener conjuntos de atributos diferentes. Esto los diferencia de los modelos de datos mencionados anteriormente, en los que todos los elementos de datos de un tipo dado deben tener el mismo conjunto de atributos.

El lenguaje XML se diseñó inicialmente como un modo de añadir información de marcas a los documentos de texto, pero se ha vuelto importante debido a sus aplicaciones en el intercambio de datos. XML ofrece un modo de representar los datos que tienen una estructura anidada y, además, permite una gran flexibilidad en la estructuración de los datos, lo cual es importante para ciertas clases de datos no tradicionales. En el Capítulo 10 se describe el lenguaje XML, diferentes maneras de expresar las consultas sobre datos representados en XML y la transformación de los datos XML de una forma a otra.

1.8 Almacenamiento de datos y consultas

Los sistemas de bases de datos están divididos en módulos que tratan con cada una de las responsabilidades del sistema general. Los componentes funcionales de los sistemas de bases de datos pueden dividirse grosso modo en los componentes gestor de almacenamiento y procesador de consultas.

El gestor de almacenamiento es importante porque las bases de datos suelen necesitar una gran cantidad de espacio de almacenamiento. Las bases de datos corporativas tienen un tamaño que va de los centenares de gigabytes hasta, para las bases de datos de mayor tamaño, los terabytes de datos. Un gigabyte son aproximadamente 1.000 megabytes (1.000 millones de bytes), y un terabyte es aproximadamente un millón de megabytes (1 billón de bytes). Debido a que la memoria principal de las computadoras no puede almacenar toda esta información, la información se almacena en discos. Los datos se intercambian entre los discos de almacenamiento y la memoria principal cuando sea necesario. Como el intercambio de datos con el disco es lento comparado 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 intercambio de datos entre los discos 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 se ven molestados innecesariamente con los detalles físicos de la implementación del sistema. Sin embargo, el rápido procesamiento de las actualizaciones y de las consultas es importante. Es función del sistema de bases de datos traducir las actualizaciones y las consultas escritas en lenguajes no procedimentales, en el nivel lógico, en una secuencia eficiente 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 almacenados en la base de datos y los programas de aplicación y las consultas remitidas al sistema. El gestor de almacenamiento es responsable de la interacción con el gestor de archivos. Los datos en bruto se almacenan en el disco mediante el sistema de archivos que suele proporcionar un sistema operativo convencional. El gestor de almacenamiento traduce las diferentes instrucciones LMD a comandos de bajo nivel del sistema de archivos. Así, el gestor de almacenamiento es responsable del almacenamiento, la recuperación y la actualización de los datos de la base de datos.

Entre los componentes del gestor de almacenamiento se encuentran:

- **Gestor de autorizaciones e integridad**, que comprueba que se satisfagan las restricciones de integridad y la autorización de los usuarios para tener acceso a los datos.

- **Gestor de transacciones**, que garantiza que la base de datos quede en un estado consistente (correcto) a pesar de los fallos del sistema, y que la ejecución concurrente de transacciones transcurra si conflictos.
- **Gestor de archivos**, que gestiona la asignación de espacio de almacenamiento de disco y las estructuras de datos usadas para representar la información almacenada en el disco.
- **Gestor de la memoria intermedia**, que es responsable de traer los datos desde el disco de almacenamiento a la memoria principal y decidir los datos a guardar en la memoria caché. El gestor de la memoria intermedia es una parte fundamental de los sistemas 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 misma.
- **Diccionario de datos**, que almacena metadatos acerca de la estructura de la base de datos; en particular, su esquema.
- **Índices**, que pueden proporcionar un acceso rápido a los elementos de datos. Como el índice de este libro de texto, los índices de las bases de datos facilitan punteros a los elementos de datos que tienen un valor concreto. Por ejemplo, se puede usar un índice para buscar todos los registros *cuenta* con un *número_cuenta* determinado. La asociación es una alternativa a la indexación que es más rápida en algunos casos, pero no en todos.

Se estudiarán los medios de almacenamiento, las estructuras de archivos y la gestión de la memoria intermedia en el Capítulo 11. Los métodos de acceso eficiente a los datos mediante indexación o asociación se explican en el Capítulo 12.

1.8.2 El procesador de consultas

Entre los componentes del procesador de consultas se encuentran:

- **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 entienda el motor de evaluación de consultas.

  Las consultas se suelen poder traducir en varios planes de ejecución alternativos, todos los cuales 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 opciones posibles.
- **Motor de evaluación de consultas**, que ejecuta las instrucciones de bajo nivel generadas por el compilador del LMD.

La evaluación de las consultas se trata en el Capítulo 13, mientras que los métodos por los que el optimizador de consultas elige entre las estrategias de evaluación posibles se tratan en el Capítulo 14.

1.9 Gestión de transacciones

A menudo, varias operaciones sobre la base de datos forman una única unidad lógica de trabajo. Un ejemplo son las transferencia de fondos, como se vio en el Apartado 1.2, en las que se realiza un cargo en una cuenta (llámese *A*) y un abono en otra cuenta (llámese *B*). Evidentemente, resulta fundamental que, o bien tengan lugar tanto el cargo como el abono, o bien que no se produzca ninguno. Es decir, la transferencia de fondos debe tener lugar por completo o no producirse en absoluto. Este requisito

de todo o nada se denomina **atomicidad**. Además, resulta 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 denomina **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 denomina **durabilidad**.

Una **transacción** es un conjunto de operaciones que lleva a cabo una única función lógica en una aplicación de bases de datos. Cada transacción es una unidad de atomicidad y consistencia. Por tanto, se exige 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, 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 el cargo a A o el abono a B se debe realizar en primer lugar. 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 puede definirse como si estuviera compuesta de dos programas diferentes: uno que realiza el cargo en la cuenta A y otro que realiza el abono en 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 a otro nuevo. Por tanto, estos programas no son transacciones.

Garantizar las propiedades de atomicidad y de durabilidad es responsabilidad del propio sistema de bases de datos —concretamente del **componente de gestión de transacciones**. A falta de fallos, todas las transacciones se completan con éxito y la atomicidad se consigue fácilmente. Sin embargo, debido a diversos tipos de fallos, puede que las transacciones no siempre completen su ejecución con éxito. Si se va a asegurar la propiedad de atomicidad, las transacciones fallidas no deben tener ningún efecto sobre el estado de la base de datos. Por tanto, la base de datos debe restaurarse al estado en que estaba antes de que la transacción en cuestión comience a ejecutarse. El sistema de bases de datos, por tanto, debe realizar la **recuperación de fallos**, es decir, detectar los fallos del sistema y restaurar la base de datos al estado que tenía antes de que ocurriera el fallo.

Finalmente, cuando varias transacciones actualizan la base de datos de manera concurrente, puede que no se preserve la consistencia de los datos, aunque cada una de las transacciones sea correcta. Es responsabilidad del **gestor de control de concurrencia** controlar la interacción entre las transacciones concurrentes para garantizar la consistencia de la base de datos.

Los conceptos básicos del procesamiento de transacciones se tratan en el Capítulo 15. La gestión de las transacciones concurrentes se trata en el Capítulo 16. En el Capítulo 17 se trata con detalle la recuperación de fallos.

Puede que los sistemas de bases de datos diseñados para su empleo en computadoras personales pequeños no tengan todas estas características. Por ejemplo, muchos sistemas pequeños sólo permiten que un usuario tenga acceso a la base de datos en cada momento. Otros no ofrecen copias de seguridad ni recuperación, y dejan esas tareas a los usuarios. Estas restricciones permiten un gestor de datos de menor tamaño, con menos requisitos de recursos físicos —especialmente de memoria principal. Aunque tales enfoques de bajo coste y bajas prestaciones son adecuados para bases de datos personales pequeñas, resultan inadecuados para empresas medianas y grandes.

El concepto de transacción se ha aplicado ampliamente en los sistemas y en las aplicaciones de bases de datos. Aunque el empleo inicial de las transacciones se produjo en las aplicaciones financieras, el concepto se usa ahora en aplicaciones de tiempo real de telecomunicaciones, así como en la gestión de las actividades de larga duración como el diseño de productos o los flujos de trabajo administrativos. Estas aplicaciones más amplias del concepto de transacción se estudian en el Capítulo 25.

1.10 Minería y análisis de datos

El término **minería de datos** se refiere en líneas generales al proceso de análisis semiautomático de grandes bases de datos para descubrir patrones útiles. Al igual que el descubrimiento de conocimiento en inteligencia artificial (también denominado **aprendizaje de la máquina**) o el análisis estadístico, la minería de datos intenta descubrir reglas y patrones en los datos. Sin embargo, la minería de datos

se diferencia del aprendizaje de la máquina y de la estadística en que maneja grandes volúmenes de datos, almacenados principalmente en disco. Es decir, la minería de datos trata del "descubrimiento de conocimiento en las bases de datos".

Algunos tipos de conocimiento descubiertos en las bases de datos pueden representarse mediante un conjunto de **reglas**. Lo que sigue es un ejemplo de regla, definida informalmente: "las mujeres jóvenes con ingresos anuales superiores a 50.000 € son las personas con más probabilidades de comprar coches deportivos pequeños". Por supuesto, esas reglas no son universalmente ciertas, sino que tienen grados de "apoyo" y de "confianza". Otros tipos de conocimiento se representan mediante ecuaciones que relacionan diferentes variables, o mediante otros mecanismos para la predicción de los resultados cuando se conocen los valores de algunas variables.

Hay gran variedad de tipos posibles de patrones que pueden resultar útiles y se emplean diferentes técnicas para descubrir tipos de patrones diferentes. En el Capítulo 18 se estudian unos cuantos ejemplos de patrones y se ve la manera en que pueden obtenerse de las bases de datos de forma automática.

Generalmente hay un componente manual en la minería de datos, que consiste en el preprocesamiento de los datos de una manera aceptable para los algoritmos, y el postprocesamiento de los patrones descubiertos para descubrir otros nuevos que puedan resultar útiles. Puede haber también más de un tipo de patrón que pueda descubrirse en una base de datos dada, y puede ser necesaria la interacción manual para escoger los tipos de patrones útiles. Por este motivo, la minería de datos es, en realidad, un proceso semiautomático en la vida real. No obstante, en nuestra descripción se concentra la atención en el aspecto automático de la minería.

Las empresas han comenzado a explotar la creciente cantidad de datos en línea para tomar mejores decisiones sobre sus actividades, como los artículos de los que hay que tener existencias y la mejor manera de llegar a los clientes para incrementar las ventas. Muchas de sus consultas son bastante complicadas, sin embargo, y ciertos tipos de información no pueden extraerse ni siquiera usando SQL.

Se dispone de varias técnicas y herramientas para ayudar a la toma de decisiones. Varias herramientas para el análisis de datos permiten a los analistas ver los datos de diferentes maneras. Otras herramientas de análisis realizan cálculos previos de resúmenes de grandes cantidades de datos, con objeto de dar respuestas rápidas a las preguntas. El estándar SQL:1999 contiene actualmente constructores adicionales para soportar el análisis de datos.

Los datos textuales también han crecido de manera explosiva. Estos datos carecen de estructura, a diferencia de los datos rígidamente estructurados de las bases de datos relacionales. La consulta de datos textuales no estructurados se denomina *recuperación de la información*. Los sistemas de recuperación de la información tienen mucho en común con los sistemas de bases de datos —en especial, el almacenamiento y recuperación de datos en medios de almacenamiento secundarios. Sin embargo, el énfasis en el campo de los sistemas de información es diferente del de los sistemas de bases de datos, y se concentra en aspectos como las consultas basadas en palabras clave; la relevancia de los documentos para la consulta, y el análisis, clasificación e indexación de los documentos. En los Capítulos 18 y 19 se trata la ayuda a la toma de decisiones, incluyendo el procesamiento analítico en línea, la minería de datos y la recuperación de la información.

1.11 Arquitectura de las bases de datos

Ahora es posible ofrecer una visión única (Figura 1.6) de los diversos componentes de los sistemas de bases de datos y de las conexiones existentes entre ellos.

La arquitectura de los sistemas de bases de datos se ve muy influida por el sistema informático subyacente sobre el que se ejecuta el sistema de bases de datos. Los sistemas de bases de datos pueden estar centralizados o ser del tipo cliente-servidor, en los que una máquina servidora ejecuta el trabajo en nombre de multitud de máquinas clientes. Los sistemas de bases de datos pueden diseñarse también para aprovechar las arquitecturas de computadoras paralelas. Las bases de datos distribuidas se extienden por varias máquinas geográficamente separadas.

En el Capítulo 20 se trata la arquitectura general de los sistemas informáticos modernos. El Capítulo 21 describe el modo en que diversas acciones de las bases de datos, en especial el procesamiento de las consultas, pueden implementarse para aprovechar el procesamiento paralelo. El Capítulo 22 presenta varios problemas que surgen en las bases de datos distribuidas y describe el modo de afrontarlos. Entre

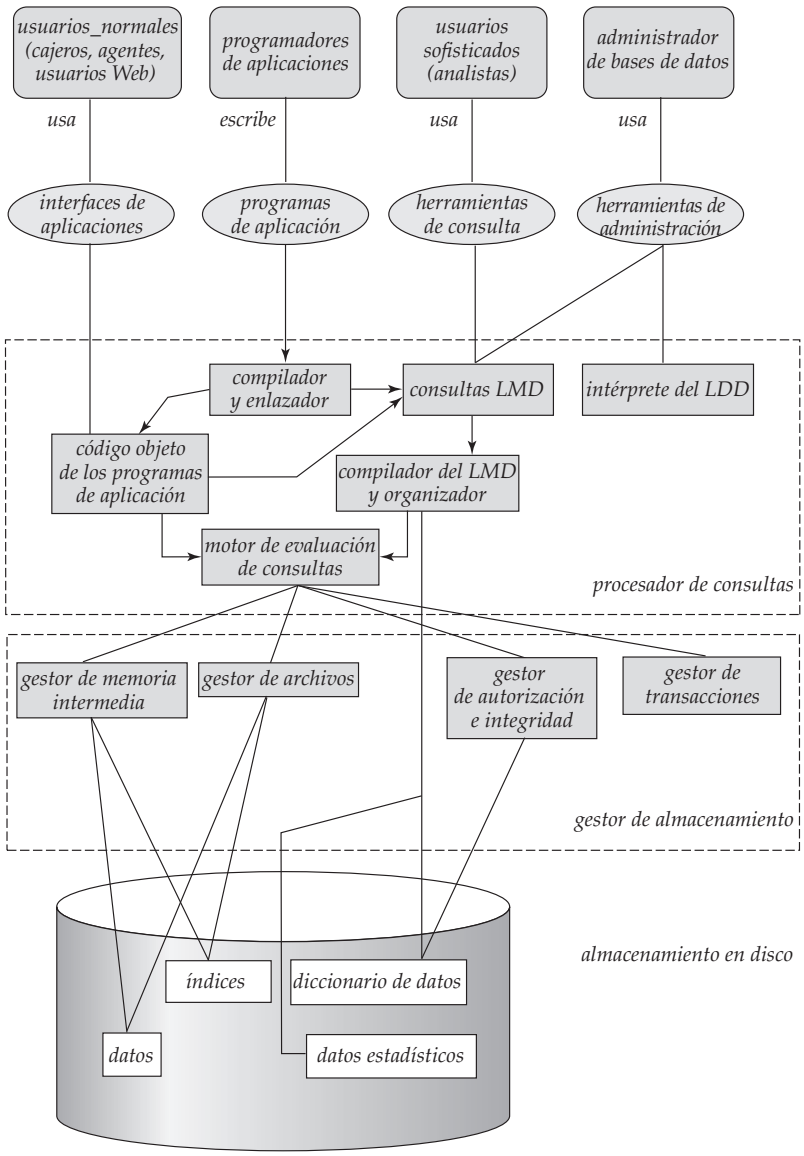


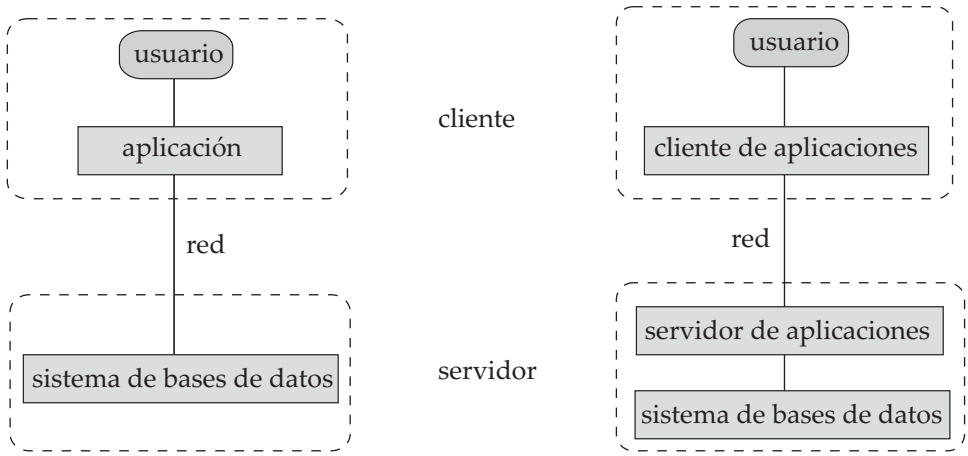
Figura 1.6 Arquitectura del sistema.

los problemas se encuentran el modo de almacenar los datos, la manera de asegurar la atomicidad de las transacciones que se ejecutan en varios sitios, cómo llevar a cabo controles de concurrencia y el modo de ofrecer alta disponibilidad en presencia de fallos. El procesamiento distribuido de las consultas y los sistemas de directorio también se describen en ese capítulo.

Hoy en día la mayor parte de los usuarios de los sistemas de bases de datos no está presente en el lugar físico en que se encuentra el sistema de bases de datos, sino que se conectan a él a través de una red. Por tanto, se puede diferenciar entre los sistemas **clientes**, en los que trabajan los usuarios remotos de la base de datos, y los sistemas **servidores**, en los que se ejecutan los sistemas de bases de datos.

Las aplicaciones de bases de datos suelen dividirse en dos o tres partes, como puede verse en la Figura 1.7. 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 servidora 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 una parte visible al usuario y no contiene ninguna llamada directa a la base de datos. En vez de eso, el extremo



(a) Arquitectura de dos capas (b) Arquitectura de tres capas

Figura 1.7 Arquitecturas de dos y tres capas.

cliente se comunica con un **servidor de aplicaciones**, generalmente mediante una interfaz de formularios. El servidor de aplicaciones, a su vez, se comunica con el sistema de bases de datos para tener acceso a los datos. La **lógica de negocio** de la aplicación, que establece las acciones que se deben realizar según las condiciones reinantes, se incorpora en el servidor de aplicaciones, en lugar de estar distribuida entre múltiples clientes. Las aplicaciones de tres capas resultan más adecuadas para aplicaciones de gran tamaño y para las aplicaciones que se ejecutan en World Wide Web.

1.12 Usuarios y administradores de bases de datos

Uno de los objetivos principales de los sistemas de bases de datos es recuperar información de la base de datos y almacenar en ella información nueva. Las personas que trabajan con una base de datos se pueden clasificar como usuarios o administradores de bases de datos.

1.12.1 Usuarios de bases de datos e interfaces de usuario

Hay cuatro tipos diferentes de usuarios de los sistemas de bases de datos, diferenciados por la forma en que esperan interactuar con el sistema. Se han diseñado diferentes tipos de interfaces de usuario para los diferentes tipos de usuarios.

- Los **usuarios normales** son usuarios no sofisticados que interactúan con el sistema invocando alguno de los programas de aplicación que se han escrito previamente. Por ejemplo, un cajero bancario que necesita transferir 50 € de la cuenta *A* a la cuenta *B* invoca un programa llamado *transferencia*. Ese programa le pide al cajero el importe de dinero que se va a transferir, la cuenta desde la que se va a transferir el dinero y la cuenta a la que se va a transferir el dinero.

  Como ejemplo adicional, considérese un usuario que desea averiguar el saldo de su cuenta en World Wide Web. Ese usuario puede 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 devuelve la información al usuario.

  La interfaz de usuario habitual para los usuarios normales es una interfaz de formularios, donde el usuario puede rellenar los campos correspondientes del formulario. Los usuarios normales también pueden limitarse a leer *informes* generados por la base de datos.

- Los **programadores de aplicaciones** son profesionales informáticos que escriben programas de aplicación. Los programadores de aplicaciones pueden elegir entre muchas herramientas para desarrollar las interfaces de usuario. Las herramientas de **desarrollo rápido de aplicaciones**

(DRA) son herramientas que permiten al programador de aplicaciones crear formularios e informes con un mínimo esfuerzo de programación.

- Los **usuarios sofisticados** interactúan con el sistema sin escribir programas. En su lugar, formulan sus consultas en un lenguaje de consultas de bases de datos. Remiten cada una de las consultas al **procesador de consultas**, cuya función es dividir las instrucciones LMD en instrucciones que el gestor de almacenamiento entienda. Los analistas que remiten las consultas para explorar los datos de la base de datos entran en esta categoría.
- Los **usuarios especializados** son usuarios sofisticados que escriben aplicaciones de bases de datos especializadas que no encajan en el marco tradicional del procesamiento de datos. Entre estas aplicaciones están los sistemas de diseño asistido por computadora, los sistemas de bases de conocimientos y los sistemas expertos, los sistemas que almacenan datos con tipos de datos complejos (por ejemplo, los datos gráficos y los datos de sonido) y los sistemas de modelado del entorno. En el Capítulo 9 se estudian varias de estas aplicaciones.

1.12.2 Administrador de bases de datos

Una de las principales razones de usar SGBDs es tener un control centralizado tanto de los datos como de los programas que tienen acceso a esos datos. La persona que tiene ese control central sobre el sistema se denomina **administrador de bases de datos** (**ABD**). Las funciones del ABD incluyen:

- La **definición del esquema**. El ABD crea el esquema original de la base de datos mediante la ejecución de un conjunto de instrucciones de definición de datos en el LDD.
- La **definición de la estructura y del método de acceso**.
- La **modificación del esquema y de la organización física**. El ABD realiza modificaciones en el esquema y en la organización física para reflejar las necesidades cambiantes de la organización, o para alterar la organización física a fin de mejorar el rendimiento.
- La **concesión de autorización para el acceso a los datos**. Mediante la concesión de diferentes tipos de autorización, el administrador de bases de datos puede regular las partes de la base de datos a las que puede tener acceso cada usuario. La información de autorización se guarda en una estructura especial del sistema que el SGBD consulta siempre que alguien intenta tener acceso a los datos del sistema.
- El **mantenimiento rutinario**. Algunos ejemplos de las actividades de mantenimiento rutinario del administrador de la base de datos son:
  - Copia de seguridad periódica de la base de datos, bien sobre cinta o sobre servidores remotos, para impedir la pérdida de datos en caso de desastres como las inundaciones.
  - Asegurarse de que se dispone de suficiente espacio libre en disco para las operaciones normales y aumentar el espacio en disco según sea necesario.
  - Supervisar los trabajos que se ejecuten en la base de datos y asegurarse de que el rendimiento no se degrade debido a que algún usuario haya remitido tareas muy costosas.

1.13 Historia de los sistemas de bases de datos

El procesamiento de datos impulsa el crecimiento de las computadoras, como lo ha hecho desde los primeros días de las computadoras comerciales. De hecho, la automatización de las tareas de procesamiento de datos precede a las computadoras. Las tarjetas perforadas, inventadas por Herman Hollerith, se emplearon a principios del siglo XX para registrar los datos del censo de Estados Unidos, y se usaron sistemas mecánicos para procesar las tarjetas y para tabular los resultados. Las tarjetas perforadas se usaron posteriormente con profusión como medio para introducir datos en las computadoras.

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

- **Años cincuenta y primeros años sesenta**: se desarrollaron las cintas magnéticas para el almacenamiento de datos. Las tareas de procesamiento de datos como la elaboración de nóminas se

automatizaron, con los datos almacenados en cintas. El procesamiento de datos consistía en leer datos de una o varias cintas y escribir datos en una nueva cinta. Los datos también se podían introducir desde paquetes de tarjetas perforadas e imprimirse en impresoras. Por ejemplo, los aumentos de sueldo se procesaban introduciendo los aumentos en las tarjetas perforadas y leyendo el paquete de cintas perforadas de manera sincronizada con una cinta que contenía los detalles principales de los salarios. Los registros debían estar en el mismo orden. Los aumentos de sueldo se añadían a los sueldos leídos de la cinta maestra y se escribían en una nueva cinta; esa nueva cinta se convertía en la nueva cinta maestra.

Las cintas (y los paquetes de tarjetas perforadas) sólo se podían leer secuencialmente, y el tamaño de datos era mucho mayor que la memoria principal; por tanto, los programas de procesamiento de datos se veían obligados a procesar los datos en un orden determinado, leyendo y mezclando datos de las cintas y de los paquetes de tarjetas perforadas.

- **Finales de los años sesenta y años setenta**: el empleo generalizado de los discos duros a finales de los años sesenta modificó en gran medida la situación del procesamiento de datos, ya que permitieron el acceso directo a los datos. La ubicación de los datos en disco no era importante, ya que se podía tener acceso a cualquier posición del disco en sólo unas decenas de milisegundos. Los datos se liberaron así de la tiranía de la secuencialidad. Con los discos pudieron crearse las bases de datos de red y las bases de datos jerárquicas, que permitieron que las estructuras de datos como las listas y los árboles pudieran almacenarse en disco. Los programadores pudieron crear y manipular estas estructuras de datos.

  El artículo histórico de Codd [1970] definió el modelo relacional y las formas no procedimentales de consultar los datos en el modelo relacional, y así nacieron las bases de datos relacionales. La simplicidad del modelo relacional y la posibilidad de ocultar completamente los detalles de implementación a los programadores resultaron realmente atractivas. Codd obtuvo posteriormente el prestigioso premio Turing de la ACM (Association of Computing Machinery, asociación de maquinaria informática) por su trabajo.

- **Años ochenta**: aunque académicamente interesante, el modelo relacional no se usó inicialmente en la práctica debido a sus inconvenientes en cuanto a rendimiento; las bases de datos relacionales no podían igualar el rendimiento de las bases de datos de red y jerárquicas existentes. Esta situación cambió con System R, un proyecto innovador del centro de investigación 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 comerciales de bases de datos relacionales, como DB2 de IBM, Oracle, Ingres y Rdb de DEC, desempeñaron un importante papel en el desarrollo de técnicas para el procesamiento eficiente de las consultas declarativas. En los primeros años ochenta las bases de datos relacionales habían llegado a ser competitivas frente a los sistemas de bases de datos jerárquicas y de red incluso en cuanto a rendimiento. Las bases de datos relacionales eran tan sencillas de usar que finalmente reemplazaron a las bases de datos jerárquicas y de red; los programadores que usaban esas bases de datos se veían obligados a tratar muchos detalles de implementación de bajo nivel y tenían que codificar sus consultas de forma procedimental. Lo que era aún más importante, tenían que tener presente el rendimiento durante el diseño de los programas, lo que suponía un gran esfuerzo. En cambio, en las bases de datos relacionales, casi todas estas tareas de bajo nivel las realiza de manera automática el sistema de bases de datos, lo que libera al programador para que se centre en el nivel lógico. Desde su obtención de liderazgo en los años ochenta, el modelo relacional ha reinado sin discusión entre todos los modelos de datos.

  Los años ochenta también fueron testigos 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.

- **Primeros años noventa**: 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 los años ochenta eran las aplicaciones de procesamiento de tran-

sacciones, que son intensivas en actualizaciones. La ayuda a la toma de decisiones y las consultas volvieron a emerger como una importante área de aplicación para las bases de datos. El uso de las herramientas para analizar grandes cantidades de datos experimentó un gran crecimiento.

En esta época muchas marcas de bases de datos introdujeron productos de bases de datos paralelas. Las diferentes marcas de bases de datos también comenzaron a añadir soporte relacional orientado a objetos a sus bases de datos.

- **Finales de los años noventa**: el principal acontecimiento fue el crecimiento explosivo de World Wide Web. Las bases de datos se implantaron mucho más ampliamente que nunca. Los sistemas de bases de datos tenían que soportar tasas de procesamiento de transacciones muy elevadas, así como una fiabilidad muy alta y tener disponibilidad 24 $\times$ 7 (disponibilidad 24 horas al día y 7 días a la semana, lo que significa que no hay momentos de inactividad debidos a actividades de mantenimiento planificadas). Los sistemas de bases de datos también tenían que soportar interfaces Web para los datos.
- **Principios del siglo XXI**: los principios del siglo XXI han sido testigos de la emergencia de XML y de su lenguaje de consultas asociado, XQuery, como nueva tecnología de las bases de datos. Todavía es pronto para decir el papel que XML desempeñará en las bases de datos futuras. En este periodo también se ha podido presenciar el crecimiento de las técnicas de "informática autónoma/administración automática" para la minimización del esfuerzo de administración.

1.14 Resumen

- Un **sistema gestor de bases de datos (SGBD)** consiste en un conjunto de datos interrelacionados y en un conjunto de programas para tener acceso a esos datos. Los datos describen una empresa concreta.
- 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 información.
- Los sistemas de bases de datos resultan ubicuos hoy en día, y la mayor parte de la gente interactúa, directa o indirectamente, con bases de datos muchas veces al día.
- 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 preocuparse de la seguridad de la información almacenada, en caso de caídas del sistema o de intentos de acceso sin autorización. Si los datos deben compartirse entre varios usuarios, el sistema debe evitar posibles resultados anómalos.
- Uno de los propósitos principales de los sistemas de bases de datos es ofrecer a los usuarios una visión abstracta de los datos. Es decir, el sistema oculta ciertos detalles de la manera en que los datos se almacenan y mantienen.
- Por debajo de la estructura de la base de datos se halla el **modelo de datos**: un conjunto de herramientas conceptuales para describir los datos, las relaciones entre los datos, la semántica de los datos y las restricciones de los datos.
- Un **lenguaje de manipulación de datos (LMD)** es un lenguaje que permite a los usuarios tener acceso a los datos o manipularlos. Los LMDs no procedimentales, que sólo necesitan que el usuario especifique los datos que necesita, sin especificar exactamente la manera de obtenerlos, se usan mucho hoy en día.
- Un **lenguaje de definición de datos (LDD)** es un lenguaje para la especificación del esquema de la base de datos y otras propiedades de los datos.
- El modelo de datos relacional es el más implantado para el almacenamiento de datos en las bases de datos. Otros modelos de datos son el modelo de datos orientado a objetos, el modelo relacional orientado a objetos y los modelos de datos semiestructurados.

- El diseño de bases de datos supone sobre todo el diseño del esquema de la base de datos. El modelo de datos entidad-relación (E-R) es un modelo de datos muy usado para el diseño de bases de datos. Proporciona una representación gráfica conveniente para ver los datos, las relaciones y las restricciones.
- Cada sistema de bases de datos tiene varios subsistemas:
  - El subsistema **gestor de almacenamiento** proporciona la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicación y las consultas remitidas al sistema.
  - El subsistema **procesador de consultas** compila y ejecuta instrucciones LDD y LMD.
- El **gestor de transacciones** garantiza que la base de datos permanezca en un estado consistente (correcto) a pesar de los fallos del sistema. El gestor de transacciones garantiza que la ejecución de las transacciones concurrentes se produzca sin conflictos.
- Las aplicaciones de bases de datos suelen dividirse en una fachada que se ejecuta en las máquinas clientes y una parte que se ejecuta en segundo plano. En las arquitecturas de dos capas la fachada se comunica directamente con una base de datos que se ejecuta en segundo plano. En las arquitecturas de tres capas la parte en segundo plano se divide a su vez en un servidor de aplicaciones y un servidor de bases de datos.
- Los usuarios de bases de datos se pueden dividir en varias clases, y cada clase de usuario suele usar un tipo diferente de interfaz para la base de datos.

Términos de repaso

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

Ejercicios prácticos

1.1 En este capítulo se han descrito varias ventajas importantes de los sistemas gestores de bases de datos. ¿Cuáles son sus dos inconvenientes?

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

1.3 Indíquense seis pasos importantes que se deben dar para configurar una base de datos para una empresa dada.

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

Ejercicios

1.5 Indíquense cuatro aplicaciones que se hayan usado que sea muy posible que utilicen un sistema de bases de datos para almacenar datos persistentes.

1.6 Indíquense cuatro diferencias significativas entre un sistema de procesamiento de archivos y un SGBD.

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

1.8 Indíquense cinco responsabilidades del sistema gestor de bases de datos. Para cada responsabilidad, explíquense los problemas que surgirían si no se asumiera esa responsabilidad.

1.9 Indíquense al menos dos razones para que los sistemas de bases de datos soporten la manipulación de datos mediante un lenguaje de consultas declarativo como SQL, en vez de limitarse a ofrecer una biblioteca de funciones de C o de C++ para llevar a cabo la manipulación de los datos.

1.10 Explíquense los problemas que causa el diseño de la tabla de la Figura 1.5.

1.11 ¿Cuáles son las cinco funciones principales del administrador de bases de datos?

Notas bibliográficas

A continuación se ofrece una relación de libros de propósito general, colecciones de artículos de investigación y sitios Web sobre bases de datos. Los capítulos siguientes ofrecen referencias a material sobre cada tema descrito en ese capítulo.

Codd [1970] es el artículo histórico que introdujo el modelo relacional.

Entre los libros de texto que tratan los sistemas de bases de datos están Abiteboul et al. [1995], Date [2003], Elmasri y Navathe [2003], O'Neil y O'Neil [2000], Ramakrishnan y Gehrke [2002], Garcia-Molina et al. [2001] 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 las bases de datos. Entre éstos se encuentran Bancilhon y Buneman [1990], Date [1986], Date [1990], Kim [1995], Zaniolo et al. [1997] y Hellerstein y Stonebraker [2005].

Un repaso 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], Bernstein et al. [1998] y Abiteboul et al. [2003]. La página inicial del grupo de interés especial de la ACM en gestión de datos (www.acm.org/sigmod) ofrece gran cantidad de información sobre la investigación en bases de datos. Los sitios Web de los fabricantes de bases de datos (véase a continuación el apartado *Herramientas*) proporciona detalles acerca de sus respectivos productos.

Herramientas

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

También hay una serie de sistemas de bases de datos gratuitos o de dominio público; los más usados son MySQL (www.mySQL.com) y PostgreSQL (www.postgreSQL.org).

Una lista más completa de enlaces a sitios Web de fabricantes y a otras informaciones se encuentra disponible en la página inicial de este libro, en http://www.mhe.es/universidad/informatica/fundamentos.