CAPÍTULO

1 INTRODUCCIÓN

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

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

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

1.1. APLICACIONES DE LOS SISTEMAS DE BASES DE DATOS

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.3. VISIÓN DE LOS DATOS

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

1.3.1. Abstracción de datos

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

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

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

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

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

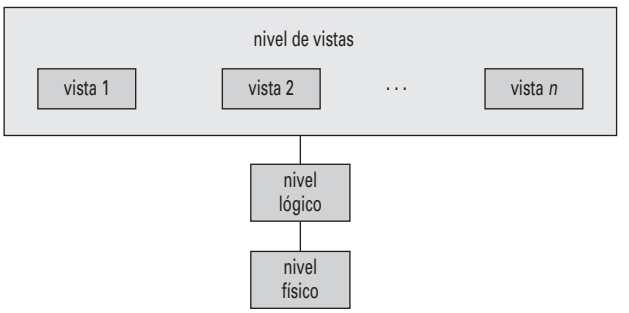

FIGURA 1.1. Los tres niveles de abstracción de datos.

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

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

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

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

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

1.3.2. Ejemplares y esquemas

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

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

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

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

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

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

1.4. MODELOS DE LOS DATOS

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

1.4.1. Modelo entidad-relación

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

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

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

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

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

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

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

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

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

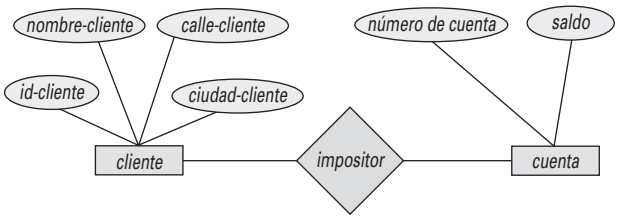

FIGURA 1.2. Ejemplo de diagrama E-R.

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

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

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

1.4.2. Modelo relacional

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

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

(a) La tabla *cliente*

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

(b) La tabla *cuenta*

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

(b) La tabla *impositor*

FIGURA 1.3. Ejemplo de base de datos relacional.

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

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

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

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

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

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

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

1.4.3. Otros modelos de datos

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

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

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

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

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

1.5. LENGUAJES DE BASES DE DATOS

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

1.5.1. Lenguaje de definición de datos

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

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

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

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

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

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

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

1.5.2. Lenguaje de manipulación de datos

La **manipulación de datos** es:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.6. USUARIOS Y ADMINISTRADORES DE LA BASE DE DATOS

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

1.6.1. Usuarios de bases de datos e interfaces de usuario

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

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

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

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

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

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

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

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

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

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

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

1.6.2. Administrador de la base de datos

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

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

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

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

1.7. GESTIÓN DE TRANSACCIONES

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

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

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

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

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

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

1.8. ESTRUCTURA DE UN SISTEMA DE BASES DE DATOS

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

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

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

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

1.8.1. Gestor de almacenamiento

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

Los componentes del gestor de almacenamiento incluyen:

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

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

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

1.8.2. Procesador de consultas

Los componentes del procesador de consultas incluyen:

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

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

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

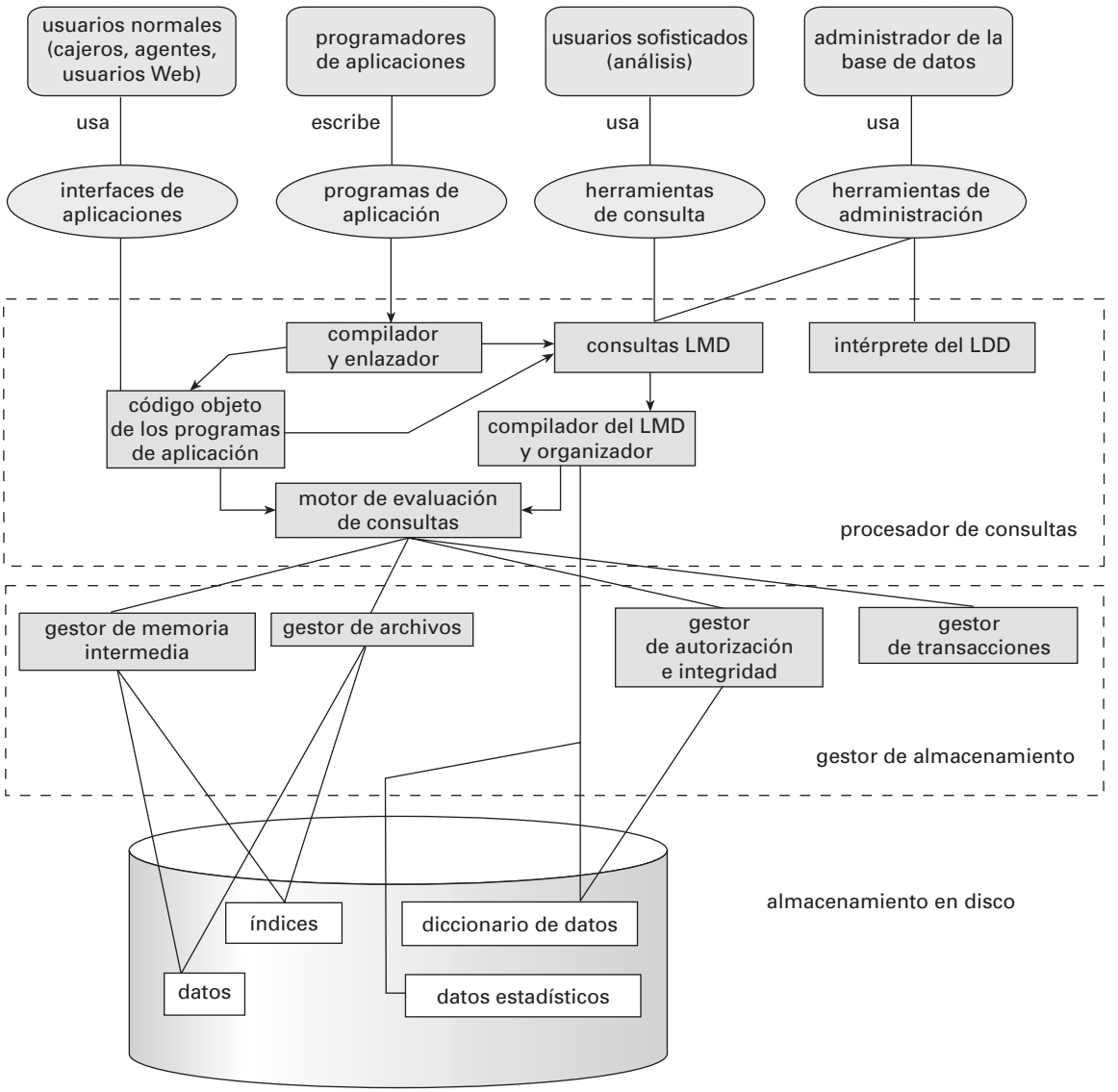

FIGURA 1.4. Estructura del sistema.

1.9. ARQUITECTURAS DE APLICACIONES

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

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

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

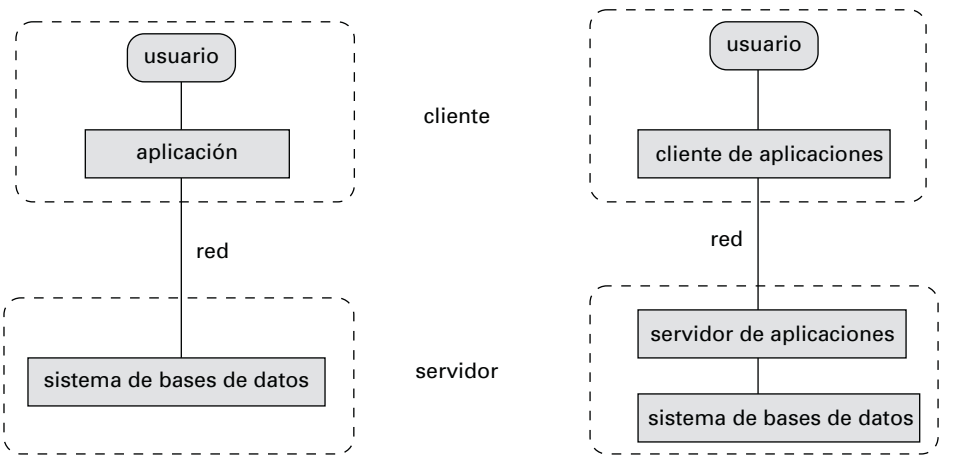

FIGURA 1.5. Arquitecturas de dos y tres capas.

1.10. HISTORIA DE LOS SISTEMAS DE BASES DE DATOS

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

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

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

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

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

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

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

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

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

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

1.11. RESUMEN

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

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

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

TÉRMINOS DE REPASO

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

EJERCICIOS

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

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

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

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

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

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

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

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