CAPÍTULO 1

Bases de datos y usuarios de bases de datos

Las bases de datos y los sistemas de bases de datos son un componente esencial de la vida cotidiana en la sociedad moderna. Actualmente, la mayoría de nosotros nos enfrentamos a diversas actividades que implican cierta interacción con una base de datos. Por ejemplo, ir al banco a depositar o retirar fondos, realizar una reserva en un hotel o una compañía aérea, acceder al catálogo computerizado de una biblioteca para buscar un libro, o comprar algo *online* (un juguete o un computador, por ejemplo), son actividades que implican que alguien o algún programa de computador acceda a una base de datos. Incluso la compra de productos en un supermercado, en muchos casos, provoca la actualización automática de la base de datos que mantiene el stock de la tienda.

Estas interacciones son ejemplos de lo que podemos llamar **aplicaciones de bases de datos tradicionales**, en las que la mayor parte de la información que hay almacenada y a la que se accede es textual o numérica. En los últimos años, los avances en la tecnología han conducido a excitantes aplicaciones y sistemas de bases de datos nuevos. La tecnología de los medios de comunicación nuevos hace posible almacenar digitalmente imágenes, clips de audio y flujos (*streams*) de vídeo. Estos tipos de archivos se están convirtiendo en un componente importante de las **bases de datos multimedia**. Los **sistemas de información geográfica (GIS, *Geographic information systems*)** pueden almacenar y analizar mapas, datos meteorológicos e imágenes de satélite. Los **almacenes de datos** y los sistemas de **procesamiento analítico en línea (OLAP**, *online analytical processing*) se utilizan en muchas compañías para extraer y analizar información útil de bases de datos mucho más grandes para permitir la toma de decisiones. Las tecnologías de **tiempo real** y **bases de datos activas** se utilizan para controlar procesos industriales y de fabricación. Y las técnicas de búsqueda en las bases de datos se están aplicando a la WWW para mejorar la búsqueda de la información que los usuarios necesitan para navegar por Internet.

No obstante, para entender los fundamentos de la tecnología de bases de datos debemos empezar por los principios básicos de las aplicaciones de bases de datos tradicionales. En la Sección 1.1 definiremos una base de datos y, a continuación, explicaremos otros términos básicos. En la Sección 1.2 ofrecemos un ejemplo de bases de datos sencillo, UNIVERSIDAD, a fin de ilustrar nuestra explicación. La Sección 1.3 describe algunas de las características principales de los sistemas de bases de datos, y las Secciones 1.4 y 1.5 clasifican los tipos de personal cuyos trabajos implican el uso e interacción con sistemas de bases de datos. Las Secciones 1.6 a 1.8 ofrecen una explicación más completa de las diferentes capacidades que los sistemas de bases de datos ofrecen y explican algunas aplicaciones de bases de datos típicas. La Sección 1.9 es un resumen del capítulo.

El lector que desee una introducción rápida a los sistemas de bases de datos sólo tiene que estudiar las Secciones 1.1 a 1.5, después omitir u ojear rápidamente las Secciones 1.6 a 1.8, y pasar al Capítulo 2.

1.1 Introducción

Las bases de datos y la tecnología de bases de datos tienen mucha culpa del uso creciente de los computadores. Es justo decir que las bases de datos juegan un papel fundamental en la mayoría de las áreas en las que se utilizan computadores, como en el ámbito empresarial, en el comercio electrónico, ingeniería, medicina, justicia, educación y bibliotecas. La expresión *base de datos* se utiliza tan a menudo que empezaremos por definir su significado. Nuestra primera definición es muy general.

Una **base de datos** es una colección de datos relacionados. Con la palabra **datos** nos referimos a los hechos (datos) conocidos que se pueden grabar y que tienen un significado implícito. Por ejemplo, piense en los nombres, números de teléfono y direcciones de las personas que conoce. Puede tener todos estos datos grabados en un libro de direcciones indexado o los puede tener almacenados en el disco duro de un computador mediante una aplicación como Microsoft Access o Excel. Esta colección de datos relacionados con un significado implícito es una base de datos.

La definición anterior de base de datos es muy genérica; por ejemplo, podemos pensar que la colección de palabras que compone esta página de texto es una colección de datos relacionados y que, por tanto, constituye una base de datos. No obstante, el uso común del término *base de datos* es normalmente más restringido. Una base de datos tiene las siguientes propiedades implícitas:

- Una base de datos representa algún aspecto del mundo real, lo que en ocasiones se denomina **minimundo** o **universo de discurso (UoD, *Universe of discourse*)**. Los cambios introducidos en el minimundo se reflejan en la base de datos.
- Una base de datos es una colección de datos lógicamente coherente con algún tipo de significado inherente. No es correcto denominar base de datos a un surtido aleatorio de datos.
- Una base de datos se diseña, construye y rellena con datos para un propósito específico. Dispone de un grupo pretendido de usuarios y algunas aplicaciones preconcebidas en las que esos usuarios están interesados.

En otras palabras, una base de datos tiene algún origen del que se derivan los datos, algún grado de interacción con eventos del mundo real y un público que está activamente interesado en su contenido. Los usuarios finales de una base de datos pueden efectuar transacciones comerciales (por ejemplo, un cliente que compra una cámara) o se pueden producir unos eventos (por ejemplo, un empleado tiene un hijo) que provoquen un cambio en la información almacenada en la base de datos. Al objeto de que una base de datos sea en todo momento precisa y fiable, debe ser un reflejo exacto del minimundo que representa; por consiguiente, en la base de datos deben reflejarse los cambios tan pronto como sea posible.

Una base de datos puede ser de cualquier tamaño y complejidad. Por ejemplo, la lista de nombres y direcciones a la que nos referíamos anteriormente puede constar de únicamente unos cuantos cientos de registros, cada uno de ellos con una estructura sencilla. Por el contrario, el catálogo computerizado de una gran biblioteca puede contener medio millón de entradas organizadas en diferentes categorías (por los apellidos del autor principal, por el tema, por el título del libro), y cada categoría ordenada alfabéticamente. El Departamento de tesorería de Estados Unidos (IRS, *Internal Revenue Service*) mantiene una base de datos de un tamaño y complejidad aún mayores para supervisar los formularios de impuestos presentados por los contribuyentes americanos. Si asumimos que hay 100 millones de contribuyentes y que cada uno presenta una media de cinco formularios con aproximadamente 400 caracteres de información por cada uno, tenemos una base de datos de $100 \times 10^6 \times 400$ x 5 caracteres (bytes) de información. Si el IRS conserva las tres últimas declaraciones de cada contribuyente, además de la actual, tenemos una base de datos de 8×10^{11} bytes (800 gigabytes). Esta

inmensa cantidad de información debe organizarse y administrarse para que los usuarios puedan buscar, recuperar y actualizar los datos que necesiten. Amazon.com es un buen ejemplo de una gran base de datos comercial. Contiene datos de más de 20 millones de libros, CDs, vídeos, DVDs, juegos, ropa y otros productos. La base de datos ocupa más de 2 terabytes (un terabyte es 10^{12} bytes de almacenamiento) y se almacena en 200 computadores diferentes (denominados servidores). Cada día acceden a Amazon.com aproximadamente 15 millones de visitantes que utilizan la base de datos para hacer compras. La base de datos se actualiza continuamente a medida que se añaden libros y otros productos nuevos al inventario, mientras que el stock se actualiza al tiempo que se tramitan las compras. Alrededor de 100 personas son las responsables de mantener actualizada la base de datos de Amazon.

Una base de datos se puede generar y mantener manualmente o estar computerizada. Por ejemplo, el catálogo de cartas de una biblioteca es una base de datos que se puede crear y mantener de forma manual. Una base de datos computerizada se puede crear y mantener con un grupo de aplicaciones escritas específicamente para esa tarea o mediante un sistema de administración de bases de datos. En este libro sólo nos ocuparemos de las bases de datos computerizadas.

Un **sistema de administración de datos (DBMS, *database management system*)** es una colección de programas que permite a los usuarios crear y mantener una base de datos. El DBMS es un *sistema de software de propósito general* que facilita los procesos de *definición*, *construcción*, *manipulación* y *compartición* de bases de datos entre varios usuarios y aplicaciones. **Definir** una base de datos implica especificar los tipos de datos, estructuras y restricciones de los datos que se almacenarán en la base de datos. La **definición** o información descriptiva de una base de datos también se almacena en esta última en forma de catálogo o diccionario de la base de datos; es lo que se conoce como **metadatos**. La **construcción** de la base de datos es el proceso consistente en almacenar los datos en algún medio de almacenamiento controlado por el DBMS. La **manipulación** de una base de datos incluye funciones como la consulta de la base de datos para recuperar datos específicos, actualizar la base de datos para reflejar los cambios introducidos en el minimundo y generar informes a partir de los datos. **Compartir** una base de datos permite que varios usuarios y programas accedan a la base de datos de forma simultánea.

Una **aplicación** accede a la base de datos enviando consultas o solicitudes de datos al DBMS. Una **consulta**[1] normalmente provoca la recuperación de algunos datos; una **transacción** puede provocar la lectura o la escritura de algunos datos en la base de datos.

Otras funciones importantes ofrecidas por el DBMS son la *protección* de la base de datos y su *mantenimiento* durante un largo periodo de tiempo. La **protección** incluye la *protección del sistema* contra el funcionamiento defectuoso del hardware o el software (caídas) y la *protección de la seguridad* contra el acceso no autorizado o malintencionado. Una gran base de datos típica puede tener un ciclo de vida de muchos años, por lo que el DBMS debe ser capaz de **mantener** el sistema de bases de datos permitiendo que el sistema evolucione según cambian los requisitos con el tiempo.

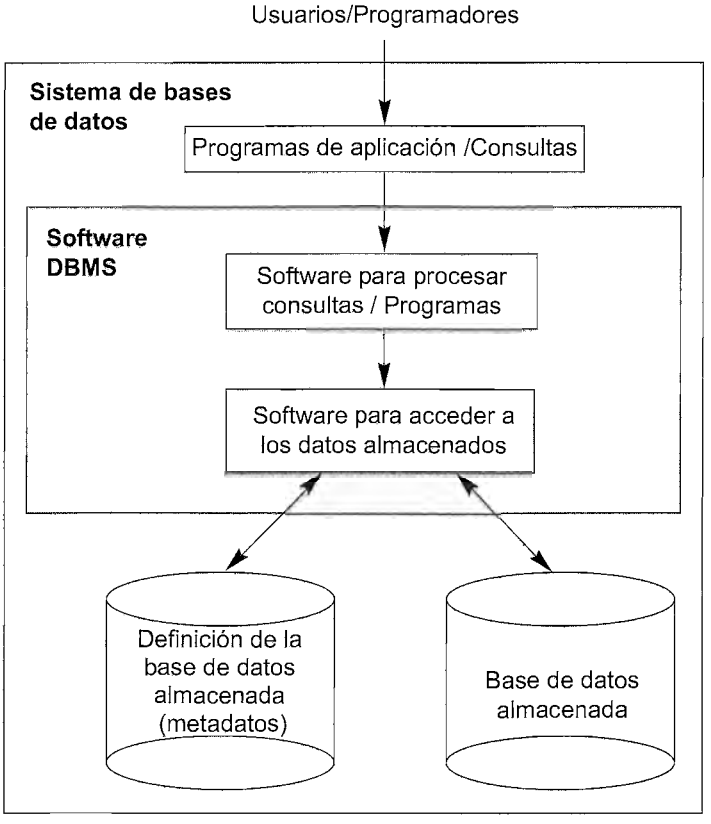
No es necesario utilizar software DBMS de propósito general para implementar una base de datos computerizada. Podríamos escribir nuestro propio conjunto de programas para crear y mantener la base de datos; en realidad, podríamos crear nuestro propio software DBMS de *propósito especial*. En cualquier caso (utilicemos o no un DBMS de propósito general), normalmente tenemos que implantar una cantidad considerable de software complejo. De hecho, la mayoría de los DBMS son sistemas de software muy complejos.

Como colofón de nuestras definiciones iniciales, denominaremos **sistema de bases de datos** a la combinación de base de datos y software DBMS. La Figura 1.1 ilustra algunos de los conceptos que hemos explicado hasta ahora.

---

[1] El término *consulta*, que inicialmente hacía referencia a una pregunta o cuestión, se utiliza ampliamente para todos los tipos de interacciones con bases de datos, incluyendo la modificación de datos.

Figura 1.1. Entorno de un sistema de bases de datos simplificado.



1.2 Un ejemplo

Vamos a ver un ejemplo con el que la mayoría de los lectores estarán familiarizados: una base de datos UNIVERSIDAD para el mantenimiento de la información relativa a los estudiantes, cursos y calificaciones en un entorno universitario. La Figura 1.2 muestra la estructura de la base de datos y algunos datos a modo de ejemplo. La base de datos está organizada en cinco archivos, cada uno de los cuales almacena registros de datos del mismo tipo.[2] El archivo ESTUDIANTE almacena los datos de todos los estudiantes, el archivo CURSO almacena los datos de todos los curso, el archivo SECCIÓN almacena los datos de las secciones de un curso, el archivo INFORME_CALIF almacena las calificaciones que los estudiantes han obtenido en las distintas secciones que han completado, y el archivo PRERREQUISITO almacena los prerrequisitos de cada curso.

Para *definir* esta base de datos debemos especificar la estructura de los registros de cada archivo detallando los diferentes tipos de **elementos de datos** que se almacenarán en cada registro. En la Figura 1.2, cada registro ESTUDIANTE incluye los datos que representan el nombre, el número, la clase (como principiante o '1', estudiante de segundo año o '2', etcétera) y la especialidad (como, por ejemplo, matemáticas o 'MAT', ciencias de la computación o 'CC'); cada registro de CURSO incluye los datos que representan el nombre, el número y las horas de crédito del curso, así como el departamento que ofrece el curso; etcétera. También hay que especificar un **tipo de datos** para cada elemento de datos de un registro. Por ejemplo, podemos especificar que el Nombre de un ESTUDIANTE es una cadena de caracteres alfabéticos, que NumEstudiante es

[2] El término *archivo* lo utilizamos aquí formalmente. A un nivel conceptual, un *archivo* es una colección de registros que pueden o no estar ordenados.

Figura 1.2. Base de datos que almacena la información de estudiantes y cursos.

ESTUDIANTE

Nombre	NumEstudiante	Clase	Especialidad
Luis	17	1	CS
Carlos	8	2	CS

CURSO

NombreCurso	NumCurso	Horas	Departamento
Introducción a la computación	CC1310	4	CC
Estructuras de datos	CC3320	4	CC
Matemáticas discretas	MAT2410	3	MAT
Bases de datos	CC3380	3	CC

SECCIÓN

IDSeccion	NumCurso	Semestre	Año	Profesor
85	MAT2410	Otoño	04	Pedro
92	CC1310	Otoño	04	Ana
102	CC3320	Primavera	05	Elisa
112	MAT2410	Otoño	05	Antonio
119	CC1310	Otoño	05	Juan
135	CC3380	Otoño	05	Enrique

INFORME_CALIF

NumEstudiante	IDSeccion	Nota
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PRERREQUISITO

NumCurso	NumPrerrequisito
CC3380	CC3320
CC3380	MAT2410
CC3320	CC1310

un entero o que la Nota de INFORME_CALIF es un solo carácter del conjunto {'A', 'B', 'C', 'D', 'F', 'I'}. También podemos utilizar un esquema de codificación para representar los valores de un elemento de datos. Por ejemplo, en la Figura 1.2 representamos la Clase de un ESTUDIANTE como 1 para los principiantes, 2 para los estudiantes de segundo año, 3 para los junior, 4 para los sénior y 5 para los estudiantes graduados.

La *construcción* de la base de datos UNIVERSIDAD se realiza almacenando los datos que representan a todos los estudiantes, cursos, secciones, informes de calificaciones y prerrequisitos a modo de registro en el archivo adecuado. Los registros de los distintos archivos se pueden relacionar. Por ejemplo, el registro correspondiente a Luis en el archivo ESTUDIANTE está relacionado con dos registros del archivo INFORME_CALIF que especifican las calificaciones de Luis en dos secciones. De forma parecida, cada registro del archivo PRERREQUISITO relaciona dos registros de curso: uno representa el curso y el otro representa el requisito previo. La mayoría de las bases de datos de medio y gran tamaño cuentan con muchos tipos de registros y tienen *muchas relaciones* entre los registros.

La *manipulación* de bases de datos implica la consulta y la actualización. A continuación tiene algunos ejemplos de consultas:

- Recuperar el certificado de estudios (listado de todos los cursos y calificaciones) de 'Luis'.
- Listado con los nombres de los estudiantes que tomaron la sección del curso 'Bases de datos' ofrecida en otoño de 2005, así como sus calificaciones en esa sección.
- Listado de los prerrequisitos del curso 'Bases de datos'.

Y estos son algunos ejemplos de actualizaciones:

- Cambiar la clase de 'Luis' a estudiante de segundo año.
- Crear una sección nueva para el curso 'Bases de datos' para este semestre.
- Introducir una nota 'A' para 'Luis' en la sección 'Bases de datos' del último semestre.

Estas consultas y modificaciones informales deben especificarse con exactitud en el lenguaje de consulta del DBMS antes de poder ser procesadas.

A estas alturas, es útil describir la base de datos como una parte de una tarea más amplia conocida como sistema de información dentro de cualquier organización. El departamento de Tecnología de la información (TI, *Information Technology*) de una empresa diseña y mantiene un sistema de información compuesto por varios computadores, sistemas de almacenamiento, aplicaciones y bases de datos. El diseño de una aplicación nueva para una base de datos existente o el diseño de una base de datos nueva empieza con una fase denominada **definición de requisitos y análisis**. Estos requisitos son documentados en detalle y transformados en un **diseño conceptual** que se puede representar y manipular mediante algunas herramientas computerizadas, de modo que en una implementación de base de datos puedan mantenerse, modificarse y transformarse fácilmente. En el Capítulo 3 introduciremos un modelo denominado Entidad-Relación que se utiliza con este propósito. El diseño después se convierte en un **diseño lógico** que se puede expresar en un modelo de datos implementado en un DBMS comercial. Del Capítulo 5 en adelante destacaremos un modelo de datos conocido como modelo de Datos relacionales. Actualmente es la metodología más popular para diseñar e implementar bases de datos utilizando DBMSs (relacionales). La etapa final es el **diseño físico**, durante la que se proporcionan especificaciones suplementarias para el almacenamiento y acceso a la base de datos. El diseño de base de datos se implementa y rellena con datos reales y se realiza un mantenimiento continuado a fin de reflejar el estado del minimundo.

1.3 Características de la metodología de bases de datos

Unas cuantas características distinguen la metodología de bases de datos de la metodología tradicional de programación con archivos. En el procesamiento tradicional de archivos, cada usuario define e implementa los

archivos necesarios para una aplicación concreta como parte de la programación de esa aplicación. Por ejemplo, un usuario, la *oficina de notificación de calificaciones*, puede encargarse del mantenimiento de un archivo con los estudiantes y sus calificaciones. Los programas encargados de imprimir el certificado de estudios de un estudiante e introducir nuevas calificaciones en el archivo se implementan como parte de la aplicación. Un segundo usuario, la *oficina de contabilidad*, puede encargarse del seguimiento de las cuotas de los estudiantes y sus pagos. Aunque ambos usuarios están interesados en datos relacionados con los estudiantes, cada uno mantiene archivos separados (y programas para la manipulación de esos archivos), porque cada uno requiere algunos datos que no están disponibles en los archivos del otro. Esta redundancia en la definición y el almacenamiento de datos da como resultado un derroche de espacio de almacenamiento y unos esfuerzos redundantes por mantener al día datos comunes.

En la metodología de bases de datos se mantiene un único almacén de datos, que se define una sola vez, y al que acceden varios usuarios. En los sistemas de archivos cada aplicación tiene libertad para asignar un nombre independientemente a los elementos de datos. Por el contrario, en una base de datos, los nombres o etiquetas de los datos se definen una vez, y son utilizados por consultas, transacciones y aplicaciones. Las principales características de la metodología de bases de datos frente a la metodología de procesamiento de archivos son las siguientes:

- Naturaleza autodescriptiva de un sistema de bases de datos.
- Aislamiento entre programas y datos, y abstracción de datos.
- Soporte de varias vistas de los datos.
- Compartición de datos y procesamiento de transacciones multiusuario.

Explicaremos cada una de estas características en una sección separada. En las Secciones 1.6 a 1.8 hablaremos de otras características adicionales de los sistemas de bases de datos.

1.3.1 Naturaleza autodescriptiva de un sistema de bases de datos

Una característica fundamental de la metodología de bases de datos es que el sistema de bases de datos no sólo contiene la propia base de datos, sino también una completa definición o descripción de la estructura de la base de datos y sus restricciones. Esta definición se almacena en el catálogo DBMS, que contiene información como la estructura de cada archivo, el tipo y el formato de almacenamiento de cada elemento de datos, y distintas restricciones de los datos. La información almacenada en el catálogo se denomina **metadatos** y describe la estructura de la base de datos principal (véase la Figura 1.1).

El software DBMS y los usuarios de la base de datos utilizan el catálogo cuando necesitan información sobre la estructura de la base de datos. Un paquete de software DBMS de propósito general no se escribe para una aplicación de base de datos específica. Por consiguiente, debe referirse al catálogo para conocer la estructura de los archivos de una base de datos específica, como el tipo y el formato de los datos a los que accederá. El software DBMS debe funcionar igual de bien con *cualquier cantidad de aplicaciones de bases de datos* (por ejemplo, la base de datos de una universidad, la base de datos de un banco o la base de datos de una empresa), siempre y cuando la definición de la base de datos esté almacenada en el catálogo.

En el procesamiento de archivos tradicional, normalmente la definición de datos forma parte de los programas de aplicación. Así pues, esas aplicaciones están restringidas a trabajar sólo con *una base de datos específica*, cuya estructura está declarada en dichas aplicaciones. Por ejemplo, una aplicación escrita en C++ puede tener declaraciones struct o class, y un programa COBOL puede tener sentencias de "*data division*" para definir sus archivos. Mientras el software de procesamiento de archivos sólo puede acceder a bases de datos específicas, el software DBMS puede acceder a distintas bases de datos extrayendo del catálogo las definiciones de las mismas y utilizando después esas definiciones.

Para el ejemplo de la Figura 1.2, el catálogo DBMS almacenará las definiciones de todos los archivos mostrados. La Figura 1.3 muestra algunas entradas de ejemplo en un catálogo de base de datos. El diseñador de

Figura 1.3. Ejemplo de catálogo de base de datos para la base de datos de la Figura 1.2.

RELACIONES

NombreRelacion	NumDeColumnas
ESTUDIANTE	4
CURSO	4
SECCIÓN	5
INFORME_CALIF	3
PRERREQUISITO	2

COLUMNAS

NombreColumna	TipoDatos	PerteneceARelacion
Nombre	Carácter (30)	ESTUDIANTE
NumEstudiante	Carácter (4)	ESTUDIANTE
Clase	Entero (1)	ESTUDIANTE
Especialidad	TipoEspecialidad	ESTUDIANTE
NombreCurso	Carácter (10)	CURSO
NumCurso	XXXXNNNN	CURSO
....	....	.....
....	....	.....
....	....	.....
NumPrerrequisito	XXXXNNNN	PRERREQUISITO

Nota: TipoEspecialidad se define como un tipo enumerado con todas las especialidades conocidas. XXXXNNNN se utiliza para definir un tipo con cuatro caracteres alfanuméricos seguidos por cuatro dígitos.

la base de datos especifica estas definiciones antes de crear la base de datos y se almacenan en el catálogo. Siempre que se crea una solicitud para acceder, por ejemplo, al Nombre de un registro de ESTUDIANTE, el software DBMS recurre al catálogo para determinar la estructura del archivo ESTUDIANTE y la posición y el tamaño del elemento de datos Nombre dentro de un registro ESTUDIANTE. Por el contrario, en una aplicación de procesamiento de archivos típica, la estructura del archivo y, en caso extremo, la ubicación exacta de Nombre dentro de un registro ESTUDIANTE también están codificadas dentro de cada programa que accede a dicho elemento de datos.

1.3.2 Aislamiento entre programas y datos, y abstracción de datos

En el procesamiento de archivos tradicional, la estructura de los archivos de datos está incrustada en las aplicaciones, por lo que los cambios que se introducen en la estructura de un archivo pueden obligar a realizar *cambios en todos los programas* que acceden a ese archivo. Por el contrario, los programas que acceden a un DBMS no necesitan esos cambios en la mayoría de los casos. La estructura de los archivos de datos se almacena en el catálogo DBMS, independientemente de los programas de acceso. Llamaremos a esta propiedad **independencia programa-datos**.

Por ejemplo, un programa de acceso a archivos puede escribirse de modo que sólo pueda acceder a los registros ESTUDIANTE de la estructura mostrada en la Figura 1.4. Si queremos añadir otra porción de datos a cada registro ESTUDIANTE, por ejemplo FechaNac, un programa semejante ya no funcionará y deberá modificarse. Por el contrario, en un entorno DBMS, sólo tendremos que cambiar la descripción de los registros ESTUDIANTE en el catálogo (véase la Figura 1.3) para reflejar la inclusión del nuevo elemento de datos FechaNac; ningún programa cambia. La siguiente vez que un programa DBMS haga referencia al catálogo, se podrá utilizar y acceder a la estructura nueva de los registros ESTUDIANTE.

En algunos tipos de sistemas de bases de datos, como los sistemas orientados a objetos y los de objetos relacionales (consulte los Capítulos 20 a 22), los usuarios pueden definir operaciones sobre los datos como parte de las definiciones de la base de datos. Una **operación** (también denominada *función* o *método*) se puede especificar de dos formas. La *interfaz* (o *firma*) de una operación incluye el nombre de la operación y los tipos de datos de sus argumentos (o parámetros). La *implementación* (o *método*) de la operación se especifica separadamente y puede modificarse sin que la interfaz se vea afectada. Las aplicaciones de usuario pueden operar sobre los datos invocando estas operaciones por sus nombres y argumentos, independientemente de cómo estén implementadas las operaciones. Esto puede recibir el nombre de **independencia programa-operación**.

La característica que permite la independencia programa-datos y la independencia programa-operación se denomina **abstracción de datos**. Un DBMS proporciona a los usuarios una **representación conceptual** de los datos que no incluye muchos de los detalles de cómo están almacenados los datos o de cómo están implementadas las operaciones. Informalmente, un **modelo de datos** es un tipo de abstracción de datos que se utiliza para proporcionar esa representación conceptual. El modelo de datos utiliza conceptos lógicos, como objetos, sus propiedades y sus relaciones, lo que para la mayoría de los usuarios es más fácil de entender que los conceptos de almacenamiento en el computador. Por ello, el modelo de datos *oculta* los detalles del almacenamiento y de la implementación que no resultan interesantes a la mayoría de los usuarios de bases de datos.

A modo de ejemplo, considere las Figuras 1.2 y 1.3. La implementación interna de un archivo puede definirse por la longitud de su registro (el número de caracteres o bytes de cada registro) y cada elemento de datos puede especificarse mediante su byte inicial dentro de un registro y su longitud en bytes. El registro ESTUDIANTE se representaría entonces como se muestra en la Figura 1.4. Pero un usuario típico de una base de datos no se preocupa por la ubicación de cada elemento de datos dentro de un registro, ni por su longitud; más bien, la preocupación del usuario está en que cuando haga una referencia al Nombre de un ESTUDIANTE quiere obtener el valor correcto. En la Figura 1.2 se ofrece una representación conceptual de los registros ESTUDIANTE. El DBMS puede ocultar a los usuarios de la base de datos muchos otros detalles de la organización del almacenamiento de los datos (como las rutas de acceso especificadas en un archivo); los detalles sobre el almacenamiento se explican en los Capítulos 13 y 14.

En la metodología de bases de datos, la estructura detallada y la organización de cada archivo se almacenan en el catálogo. Los usuarios de bases de datos y los programas de aplicación hacen referencia a la representación conceptual de los archivos y, cuando los módulos de acceso al archivo DBMS necesita detalles sobre el almacenamiento del archivo, el DBMS los extrae del catálogo. Se pueden utilizar muchos modelos de datos para proporcionar esta abstracción de datos a los usuarios de bases de datos. Una buena parte de este libro está

Figura 1.4. Formato de almacenamiento interno de un registro ESTUDIANTE, basándose en el catálogo de la base de datos de la Figura 1.3.

Nombre del elemento de datos	Posición inicial en el registro	Longitud en caracteres (bytes)
Nombre	1	30
NumEstudiante	31	4
Clase	35	1
Especialidad	36	4

dedicada a presentar distintos modelos de datos y los conceptos que utilizan para abstraer la representación de los datos.

En las bases de datos orientadas a objetos y de objetos relacionales, el proceso de abstracción no sólo incluye la estructura de datos, sino también las operaciones sobre los datos. Estas operaciones proporcionan una abstracción de las actividades del minimundo normalmente entendidas por los usuarios. Por ejemplo, se puede aplicar una operación CALCULAR_CM a un objeto ESTUDIANTE para calcular la calificación media. Dichas operaciones pueden ser invocadas por las consultas del usuario o por las aplicaciones, sin necesidad de conocer los detalles de cómo están implementadas esas operaciones. En este sentido, una abstracción de la actividad del minimundo queda a disposición de los usuarios como una **operación abstracta**.

1.3.3 Soporte de varias vistas de los datos

Normalmente una base de datos tiene muchos usuarios, cada uno de los cuales puede necesitar una perspectiva o vista diferente de la base de datos. Una **vista** puede ser un subconjunto de la base de datos o puede contener **datos virtuales** derivados de los archivos de la base de datos pero que no están explícitamente almacenados. Algunos usuarios no tienen la necesidad de preocuparse por si los datos a los que se refieren están almacenados o son derivados. Un DBMS multiusuario cuyos usuarios tienen variedad de diferentes aplicaciones debe ofrecer facilidades para definir varias vistas. Por ejemplo, un usuario de la base de datos de la Figura 1.2 puede estar interesado únicamente en acceder e imprimir el certificado de estudios de cada estudiante; la Figura 1.5(a) muestra la vista para este usuario. Un segundo usuario, que sólo está interesado en comprobar que los estudiantes cumplen con todos los prerrequisitos de cada curso para poder registrarse, puede requerir la vista representada en la Figura 1.5(b).

Figura 1.5. Dos vistas derivadas de la base de datos de la Figura 1.2. (a) Vista del certificado de estudios. (b) Vista de los prerrequisitos del curso.

CERTIFICADO

NombreEstudiante	CertificadoEstudiante				
	NumCurso	Nota	Semestre	Año	IDSeccion
Luis	CC1310	C	Otoño	05	119
	MAT2410	B	Otoño	05	112
Carlos	MAT2410	A	Otoño	04	85
	CC1310	A	Otoño	04	92
	CC3320	B	Primavera	05	102
	CC3380	A	Otoño	05	135

(a)

PRERREQUISITO_CURSO

NombreCurso	NumCurso	Prerrequisitos
Bases de datos	CC3380	CC3320
		MAT2410
Estructuras de datos	CC3320	CC1310

(b)

1.3.4 Compartición de datos y procesamiento de transacciones multiusuario

Un DBMS multiusuario, como su nombre indica, debe permitir que varios usuarios puedan acceder a la base de datos al mismo tiempo. Esto es esencial si los datos destinados a varias aplicaciones serán integrados y mantenidos en una sola base de datos. El DBMS debe incluir software de **control de la concurrencia** para que esos varios usuarios que intentan actualizar los mismos datos, lo hagan de un modo controlado para que el resultado de la actualización sea correcto. Por ejemplo, si varios agentes de viajes intentan reservar un asiento en un vuelo, el DBMS debe garantizar que en cada momento sólo un agente tiene acceso a la asignación de ese asiento para un pasajero. Estos tipos de aplicaciones se denominan, por lo general, aplicaciones de **procesamiento de transacciones en línea (OLTP, *online transaction processing*)**. Un papel fundamental del software DBMS multiusuario es garantizar que las transacciones concurrentes operan correcta y eficazmente.

El concepto de transacción es cada vez más importante para las aplicaciones de bases de datos. Una transacción es un *programa en ejecución* o *proceso* que incluye uno o más accesos a la base de datos, como la lectura o la actualización de los registros de la misma. Se supone que una transacción ejecuta un acceso lógicamente correcto a la base de datos si lo ejecutó íntegramente sin interferencia de otras transacciones. El DBMS debe implementar varias propiedades de transacción. La propiedad **aislamiento** garantiza que parezca que cada transacción se ejecuta de forma aislada de otras transacciones, aunque puedan estar ejecutándose cientos de transacciones al mismo tiempo. La propiedad de **atomicidad** garantiza que se ejecuten o todas o ninguna de las operaciones de bases de datos de una transacción. En la Parte 5 se explican las transacciones más en profundidad.

Las características anteriores son muy importantes para distinguir un DBMS del software de procesamiento de archivos tradicional. En la Sección 1.6 explicamos las características adicionales que caracterizan un DBMS. No obstante, en primer lugar clasificaremos los diferentes tipos de personas que trabajan en el entorno de un sistema de bases de datos.

1.4 Actores de la escena

En el caso de una base de datos personal pequeña, como la lista de direcciones mencionada en la Sección 1.1, un usuario normalmente define, construye y manipula la base de datos, de modo que no se comparten datos. Sin embargo, en empresas grandes, muchas personas están implicadas en el diseño, uso y mantenimiento de una base de datos grande con cientos de usuarios. En esta sección identificamos las personas cuyos trabajos implican el uso diario de una base de datos grande; las denominaremos *actores de la escena*. En la Sección 1.5 hablaremos de las personas que podríamos llamar *trabajadores entre bambalinas* (los que trabajan en el mantenimiento del entorno del sistema de bases de datos pero que no están activamente interesados en la propia base de datos).

1.4.1 Administradores de las bases de datos

En cualquier empresa donde muchas personas utilizan los mismo recursos, se necesita un administrador jefe que supervise y administre esos recursos. En un entorno de bases de datos, el recurso principal es la base de datos en sí misma, mientras que el recurso secundario es el DBMS y el software relacionado. La administración de estos recursos es responsabilidad del **administrador de la base de datos (DBA, *database administrator*)**. El DBA es responsable del acceso autorizado a la base de datos, de la coordinación y monitorización de su uso, y de adquirir los recursos software y hardware necesarios. El DBA también es responsable de problemas como las brechas de seguridad o de unos tiempos de respuesta pobres. En las empresas grandes, el DBA está asistido por un equipo de personas que llevan a cabo estas funciones.

1.4.2 Diseñadores de las bases de datos

Los **diseñadores de las bases de datos** son los responsables de identificar los datos que se almacenarán en la base de datos y de elegir las estructuras apropiadas para representar y almacenar esos datos. Estas tareas se acometen principalmente antes de implementar y rellenar la base de datos. Es responsabilidad de los diseñadores comunicarse con todos los presuntos usuarios de la base de datos para conocer sus requisitos, a fin de crear un diseño que satisfaga sus necesidades. En muchos casos, los diseñadores forman parte de la plantilla del DBA y se les pueden asignar otras responsabilidades una vez completado el diseño de la base de datos. Estos diseñadores normalmente interactúan con los grupos de usuarios potenciales y desarrollan **vistas** de la base de datos que satisfacen los requisitos de datos y procesamiento de esos grupos. Cada vista se analiza después y se *integra* con las vistas de los otros grupos de usuarios. El diseño final de la base de datos debe ser capaz de soportar los requisitos de todos los grupos de usuarios.

1.4.3 Usuarios finales

Los usuarios finales son las personas cuyos trabajos requieren acceso a la base de datos para realizar consultas, actualizaciones e informes; la base de datos existe principalmente para ser utilizada. Los usuarios finales se pueden clasificar en varias categorías:

- Los **usuarios finales casuales** acceden ocasionalmente a la base de datos, pero pueden necesitar una información diferente en cada momento. Utilizan un sofisticado lenguaje de consulta de bases de datos para especificar sus peticiones y normalmente son administradores de nivel medio o alto u otros usuarios interesados.
- Los **usuarios finales principiantes** o **paramétricos** constituyen una parte considerable de los usuarios finales de las bases de datos. Su labor principal gira entorno a la consulta y actualización constantes de la base de datos, utilizando tipos de consultas y actualizaciones estándar (denominadas **transacciones enlatadas**) que se han programado y probado cuidadosamente. Las tareas que estos usuarios llevan a cabo son variadas:
  - Los cajeros bancarios comprueban los balances de cuentas, así como las retiradas y los depósitos de fondos.
  - Los agentes de viajes que reservan en aerolíneas, hoteles y compañías de alquiler de automóviles comprueban la disponibilidad de una solicitud dada y hacen la reserva.
  - Los empleados de las estaciones receptoras de las compañías navieras introducen las identificaciones de los paquetes mediante códigos de barras y demás información descriptiva a través de botones para actualizar una base de datos central de paquetes recibidos y en tránsito.
- Entre los **usuarios finales sofisticados** se encuentran los ingenieros, los científicos, los analistas comerciales y otros muchos que están completamente familiarizados con el DBMS a fin de implementar sus aplicaciones y satisfacer sus complejos requisitos.
- Los **usuarios finales independientes** mantienen bases de datos personales utilizando paquetes de programas confeccionados que proporcionan unas interfaces fáciles de usar y basadas en menús o gráficos. Un ejemplo es el usuario de un paquete de impuestos que almacena sus datos financieros personales de cara a la declaración de la renta.

Un DBMS típico proporciona muchas formas de acceder a una base de datos. Los usuarios finales principiantes tienen que aprender muy poco sobre los servicios del DBMS; simplemente tienen que familiarizarse con las interfaces de usuario de las transacciones estándar diseñadas e implementadas para su uso. Los usuarios casuales sólo se aprenden unos cuantos servicios que pueden utilizar repetidamente. Los usuarios sofisticados intentan aprender la mayoría de los servicios del DBMS para satisfacer sus complejos requisitos. Los usuarios independientes normalmente llegan a ser expertos en un paquete de software específico.

1.4.4 Analistas de sistemas y programadores de aplicaciones (ingenieros de software)

Los analistas de sistemas determinan los requisitos de los usuarios finales, especialmente de los usuarios finales principiantes y paramétricos, así como las especificaciones de desarrollo para las transacciones enlatadas que satisfacen esos requisitos. Los **programadores de aplicaciones** implementan esas especificaciones como programas; después, verifican, depuran, documentan y mantienen esas transacciones enlatadas. Dichos analistas y programadores (normalmente conocidos como **desarrolladores de software** o **ingenieros de software**) deben familiarizarse con todas las posibilidades proporcionadas por el DBMS al objeto de desempeñar sus tareas.

1.5 Trabajadores entre bambalinas

Además de los que diseñan, utilizan y administran una base de datos, hay otros usuarios que están asociados con el diseño, el desarrollo y el funcionamiento de un *entorno de software y sistema DBMS*. Estas personas normalmente no están interesadas en la base de datos propiamente dicha. Los denominaremos *trabajadores entre bambalinas* y los dividiremos en las siguientes categorías:

- **Diseñadores e implementadores de sistemas DBMS.** Diseñan e implementan los módulos y las interfaces DBMS como un paquete software. Un DBMS es un sistema software muy complejo compuesto por muchos componentes, o **módulos**, incluyendo los destinados a implementar el catálogo, procesar el lenguaje de consulta, procesar la interfaz, acceder y almacenar los datos en un búfer, controlar la concurrencia, y manipular la recuperación y la seguridad de los datos. El DBMS debe interactuar con otro software de sistema, como el sistema operativo y los compiladores de diversos lenguajes de programación.
- **Desarrolladores de herramientas.** Diseñan e implementan **herramientas** (paquetes de software que facilitan el modelado y el diseño de la base de datos, el diseño del sistema de bases de datos y la mejora del rendimiento). Las herramientas son paquetes opcionales que a menudo se compran por separado. Entre ellas podemos citar los paquetes para el diseño de bases de datos, la monitorización del rendimiento, las interfaces gráficas o en otros idiomas, el prototipado, la simulación y la generación de datos de prueba. En muchos casos, los fabricantes de software independiente desarrollan y comercializan estas herramientas.
- **Operadores y personal de mantenimiento** (personal de administración del sistema). Son los responsables de la ejecución y el mantenimiento real del entorno hardware y software para el sistema de bases de datos.

Aunque estas categorías de trabajadores entre bambalinas se encargan de que el sistema de bases de datos esté disponible para los usuarios finales, normalmente no utilizan la base de datos para sus propios fines.

1.6 Ventajas de utilizar una metodología DBMS

En esta sección explicaremos algunas de las ventajas de utilizar un DBMS y las capacidades que un buen DBMS debe poseer. Estas capacidades se añaden a las cuatro características principales explicadas en la Sección 1.3. El DBA debe utilizar estas capacidades para acometer una variedad de objetivos relacionados con el diseño, la administración y el uso de una base de datos multiusuario grande.

1.6.1 Control de la redundancia

En el desarrollo tradicional de software que hace uso del procesamiento de archivos, cada grupo de usuarios mantiene sus propios archivos para manipular sus aplicaciones de procesamiento de datos. Por ejemplo,

vamos a retomar la base de datos UNIVERSIDAD de la Sección 1.2; aquí, el personal que registra los cursos y la oficina de contabilidad podrían ser los dos grupos de usuarios. En la metodología tradicional, cada grupo mantiene sus propios archivos de estudiantes. La oficina de contabilidad guarda datos sobre el registro y la información de facturación relacionada, mientras que la oficina de registro hace un seguimiento de los cursos y las calificaciones de los estudiantes. Aparte de estos dos grupos, puede haber otros que dupliquen parte o todos estos mismos datos en sus archivos propios.

La **redundancia** resultante de almacenar los mismos datos varias veces conduce a serios problemas. En primer lugar, las actualizaciones lógicas sencillas (como la introducción de los datos de un estudiante nuevo) hay que hacerlas varias veces: una por cada archivo donde se almacenen los datos de los estudiantes. Esto lleva a una *duplicación del esfuerzo*. En segundo lugar, *se derrocha espacio de almacenamiento* al guardar repetidamente los mismos datos, y este problema puede llegar a ser muy serio en las bases de datos más grandes. En tercer lugar, los archivos que representan los mismos datos pueden acabar siendo incoherentes, lo que puede ocurrir cuando una determinada actualización se aplica a unos archivos y a otros no. Incluso si una actualización (por ejemplo, la adición de un estudiante nuevo) se aplica a todos los archivos adecuados, los datos relacionados con ese estudiante pueden ser *incoherentes* porque las actualizaciones han sido aplicadas por los distintos grupos de usuarios. Por ejemplo, un grupo de usuarios puede introducir erróneamente la fecha de nacimiento del estudiante ('19-ENE-1988'), mientras que otro grupo la introduce correctamente ('29-ENE-1988').

En la metodología de bases de datos, las vistas de los diferentes grupos de usuarios se integran durante el diseño de la base de datos. Idealmente, debemos tener un diseño que almacene cada elemento de datos lógico (como el nombre o la fecha de nacimiento del estudiante) *sólo en un lugar* de la base de datos. Este hecho garantiza la coherencia y ahorra espacio de almacenamiento. Sin embargo, en la práctica, a veces es necesario recurrir a una **redundancia controlada** para mejorar el rendimiento de las consultas. Por ejemplo, podemos almacenar NombreEstudiante y NumCurso de forma redundante en un archivo INFORME_CALIF (véase la Figura 1.6[a]) porque siempre que recuperemos un registro de este último, queremos recuperar el nombre del estudiante y el número del curso, junto con la calificación, el número de estudiante y el identificador de la sección. Al colocar todos los datos juntos, no tenemos que buscar en varios archivos para recopilarlos. En estos casos, el DBMS debe tener la capacidad de *controlar* esta redundancia para evitar las incoherencias entre

Figura 1.6. Almacenamiento redundante de NombreEstudiante y NumCurso en INFORME_CALIF. (a) Datos coherentes. (b) Registro incoherente.

(a) INFORME_CALIF

NumEstudiante	NombreEstudiante	IDSeccion	NumCurso	Nota
17	Luis	112	MAT2410	B
17	Luis	119	CC1310	C
8	Carlos	85	MAT2410	A
8	Carlos	92	CC1310	A
8	Carlos	102	CC3320	B
8	Carlos	135	CC3380	A

(b) INFORME_CALIF

NumEstudiante	NombreEstudiante	IDSeccion	NumCurso	Nota
17	Carlos	112	MAT2410	B

archivos. Esto se puede hacer automáticamente comprobando que los valores NombreEstudiante-NumEstudiante de cualquier registro de INFORME_CALIF de la Figura 1.6(a) coincide con alguno de los valores Nombre-NumEstudiante del registro ESTUDIANTE (véase la Figura 1.2). De forma parecida, los valores IDSeccion-NumCurso de INFORME_CALIF pueden compararse con los registros de SECCIÓN. Estas comprobaciones pueden especificarse en el DBMS durante el diseño de la base de datos y que el DBMS las ejecute automáticamente siempre que se actualice el archivo INFORME_CALIF. La Figura 1.6(b) muestra un registro de INFORME_CALIF que es incoherente con el archivo ESTUDIANTE de la Figura 1.2, que puede introducirse incorrectamente de *no controlarse la redundancia.*

1.6.2 Restricción del acceso no autorizado

Cuando varios usuarios comparten una base de datos grande, es probable que la mayoría de los mismos no tengan autorización para acceder a toda la información de la base de datos. Por ejemplo, los datos financieros se consideran a menudo confidenciales, y sólo las personas autorizadas pueden acceder a ellos. Además, algunos usuarios sólo pueden recuperar datos, mientras que otros pueden recuperarlos y actualizarlos. Así pues, también hay que controlar el tipo de operación de acceso (recuperación o actualización). Normalmente, los usuarios o grupos de usuarios tienen números de cuenta protegidos mediante contraseñas, que pueden utilizar para tener acceso a la base de datos. Un DBMS debe proporcionar **seguridad y un subsistema de autorización**, que el DBA utiliza para crear cuentas y especificar las restricciones de las mismas. Después, el DBMS debe implementar automáticamente esas restricciones. Podemos aplicar controles parecidos al software DBMS. Por ejemplo, sólo el personal del DBA puede utilizar cierto **software privilegiado**, como el que permite crear cuentas nuevas. De forma parecida, los usuarios paramétricos pueden acceder a la base de datos sólo a través de transacciones enlatadas desarrolladas para su uso.

1.6.3 Almacenamiento persistente para los objetos del programa

Las bases de datos se pueden utilizar para proporcionar **almacenamiento persistente** a los objetos de programa y las estructuras de datos. Es una de las principales razones de los **sistemas de bases de datos orientados a objetos**. Normalmente, los lenguajes de programación tienen estructuras de datos complejas, como tipos de registro en Pascal o definiciones de clase en C++ o Java. Los valores de las variables de un programa se descartan una vez que termina ese programa, a menos que el programador los almacene explícitamente en archivos permanentes, lo que a menudo implica convertir esas estructuras complejas en un formato adecuado para el almacenamiento del archivo. Cuando surge la necesidad de leer estos datos una vez más, el programador debe convertir el formato del archivo a la estructura variable del programa. Los sistemas de bases de datos orientados a objetos son compatibles con lenguajes de programación como C++ y Java, y el software DBMS realiza automáticamente las conversiones necesarias. Por tanto, un objeto complejo de C++ se puede almacenar de forma permanente en un DBMS orientado a objetos. Se dice que dicho objeto es **persistente**, porque sobrevive a la terminación de la ejecución del programa y otro programa C++ lo puede recuperar más tarde.

El almacenamiento persistente de objetos de programas y estructuras de datos es una función importante de los sistemas de bases de datos. Los sistemas de bases de datos tradicionales a menudo adolecían de lo que se denominó **problema de incompatibilidad de impedancia**, puesto que las estructuras de datos proporcionadas por el DBMS eran incompatibles con las estructuras de datos del lenguaje de programación. Los sistemas de bases de datos orientados a objetos normalmente ofrecen la **compatibilidad** de la estructura de datos con uno o más lenguajes de programación orientados a objetos.

1.6.4 Suministro de estructuras de almacenamiento para un procesamiento eficaz de las consultas

Los sistemas de bases de datos deben proporcionar capacidades para *ejecutar eficazmente consultas y actualizaciones*. Como la base de datos normalmente se almacena en el disco, el DBMS debe proporcionar estruc-

turas de datos especializadas para acelerar la búsqueda en el disco de los registros deseados. Con este fin se utilizan unos archivos auxiliares denominados **índices**, que están basados casi siempre en el árbol de estructuras de datos o en las estructuras de datos dispersas, convenientemente modificados para la búsqueda en disco. A fin de procesar los registros necesarios de la base de datos para una consulta en particular, estos registros deben copiarse del disco a la memoria. Por consiguiente, el DBMS a menudo tiene un módulo de búfer que mantiene partes de la base de datos en los búferes de la memoria principal. En otros casos, el DBMS puede utilizar el sistema operativo para realizar el volcado de los datos del disco en el búfer.

El **módulo de procesamiento y optimización de consultas** del DBMS es el responsable de elegir un plan eficaz de ejecución de consultas para cada consulta basándose en las estructuras de almacenamiento existentes. La elección de qué índices crear y mantener es parte del diseño y refinamiento de la base de datos física, que es una de las responsabilidades del personal del DBA. En los Capítulos 15 y 16 explicaremos en profundidad el procesamiento, la optimización y el refinamiento de las consultas.

1.6.5 Copia de seguridad y recuperación

Un DBMS debe ofrecer la posibilidad de recuperarse ante fallos del hardware o del software. El **subsistema de copia de seguridad y recuperación** del DBMS es el responsable de la recuperación. Por ejemplo, si el computador falla en medio de una transacción compleja de actualización, el subsistema de recuperación es responsable de garantizar la restauración de la base de datos al estado anterior a que comenzase la ejecución de la transacción. Como alternativa, el subsistema de recuperación podría asegurarse de retomar la transacción en el punto en que se interrumpió para que todo su efecto se grabe en la base de datos.

1.6.6 Suministro de varias interfaces de usuario

Como una base de datos la utilizan muchos tipos de usuarios con distintos niveles de conocimiento técnico, un DBMS debe proporcionar distintas interfaces de usuario, entre las que podemos citar los lenguajes de consulta para los usuarios casuales, las interfaces de lenguaje de programación para los programadores de aplicaciones, formularios y códigos de comando para los usuarios paramétricos, e interfaces por menús y en el idioma nativo para los usuarios independientes. Tanto las interfaces al estilo de los formularios como las basadas en menús se conocen normalmente como **interfaces gráficas de usuario (GUI, *graphical user interfaces*)**. Existen muchos entornos y lenguajes especializados para especificar las GUIs. También son muy comunes las capacidades de proporcionar interfaces GUI web a una base de datos.

1.6.7 Representación de relaciones complejas entre los datos

Una base de datos puede incluir numerosas variedades de datos que se interrelacionan entre sí de muchas formas. Considerando el ejemplo de la Figura 1.2, el registro de 'Carlos' del archivo ESTUDIANTE está relacionado con cuatro registros del archivo INFORME_CALIF. Del mismo modo, cada registro de sección está relacionado con un registro de curso y con varios registros de INFORME_CALIF (uno por cada estudiante que haya completado esa sección). Un DBMS debe tener la capacidad de representar las relaciones complejas entre los datos, definir las nuevas relaciones que surgen, y recuperar y actualizar fácil y eficazmente los datos relacionados.

1.6.8 Implementación de las restricciones de integridad

La mayoría de las aplicaciones de bases de datos tienen ciertas **restricciones de integridad** que deben mantenerse para los datos. Un DBMS debe proporcionar servicios para definir e implementar esas restricciones. El tipo de restricción de integridad más simple consiste en especificar un tipo de datos por cada elemento de datos. Por ejemplo, en la Figura 1.3 especificamos que el valor del elemento de datos Clase dentro de cada

registro ESTUDIANTE debe ser un dígito entero, y que el valor de Nombre debe ser una cadena de no más de 30 caracteres alfanuméricos. Para restringir el valor de Clase entre 1 y 5 debe haber una restricción adicional que no se muestra en el catálogo actual. Un tipo de restricción más compleja que se da a menudo implica especificar que un registro de un archivo debe estar relacionado con registros de otros archivos. Por ejemplo, en la Figura 1.2 podemos especificar que *cada registro de sección debe estar relacionado con un registro de curso*. Otro tipo de restricción especifica la unicidad en los valores del elemento de datos, como que *cada registro de curso debe tener un único valor para NumCurso*. Estas restricciones se derivan del significado o la **semántica** de los datos y del minimundo que representan. Los diseñadores tienen la responsabilidad de identificar las restricciones de integridad durante el diseño de la base de datos. Algunas restricciones pueden especificarse en el DBMS e implementarse automáticamente. Otras restricciones pueden tener que ser comprobadas por los programas de actualización o en el momento de introducir los datos. En las aplicaciones grandes es costumbre denominar estas restricciones como **reglas de negocio**.

Aun cuando se introduce erróneamente un elemento de datos, éste puede satisfacer las restricciones de integridad especificadas. Por ejemplo, si un estudiante recibe una calificación de 'A' pero se introduce una calificación de 'C' en la base de datos, el DBMS *no puede* descubrir automáticamente este error porque 'C' es un valor correcto para el tipo de datos Nota. Estos errores en la introducción de los datos sólo se pueden descubrir manualmente (cuando el estudiante recibe la calificación y reclama) y corregirse más tarde mediante la actualización de la base de datos. No obstante, una calificación de 'Z' debería rechazarla automáticamente el DBMS, porque no se trata de un valor correcto para el tipo de datos Nota. Cuando expliquemos cada modelo de datos en los siguientes capítulos, introduciremos reglas que pertenecen implícitamente a ese modelo. Por ejemplo, en el modelo Entidad-Relación del Capítulo 3, una relación debe implicar como mínimo a dos entidades. Estas reglas son **reglas inherentes** del modelo de datos y se asumen automáticamente para garantizar la validez del modelo.

1.6.9 Inferencia y acciones usando reglas

Algunos sistemas de bases de datos ofrecen la posibilidad de definir *reglas de deducción* para *inferir* información nueva a partir de los hechos guardados en la base de datos. Estos sistemas se denominan **sistemas de bases de datos deductivos**. Por ejemplo, puede haber reglas complejas en la aplicación del minimundo para determinar si un estudiante está a prueba. Éstas se pueden especificar *declarativamente* como **reglas**, de modo que cuando el DBMS las compila y mantiene pueden determinar todos los estudiantes que están en periodo de prueba. En un DBMS tradicional habría que escribir un *código de programa procedimental* explícito para soportar dichas aplicaciones. Pero si cambian las reglas del minimundo, generalmente es mejor cambiar las reglas de deducción declaradas que volver a codificar los programas procedurales. En los sistemas de bases de datos relacionales actuales es posible asociar *triggers* a las tablas. Un ***trigger*** es una forma de regla que se activa con las actualizaciones de la tabla, lo que conlleva la ejecución de algunas operaciones adicionales sobre otras tablas, el envío de mensajes, etcétera. Los procedimientos más implicados en la implementación de reglas se conocen popularmente como **procedimientos almacenados**; se convierten en parte de la definición global de la base de datos y se les invoca correctamente cuando se dan ciertas condiciones. Los **sistemas de bases de datos activos** ofrecen la funcionalidad más potente; estos sistemas proporcionan reglas activas que pueden iniciar automáticamente acciones cuando ocurren ciertos eventos y condiciones.

1.6.10 Implicaciones adicionales de utilizar la metodología de bases de datos

Esta sección explica algunas implicaciones adicionales de usar la metodología de bases de datos que pueden beneficiar a la mayoría de las empresas.

Potencial para implementar estándares. La metodología de bases de datos permite al DBA definir e implementar estándares entre los usuarios de la base de datos en una empresa grande. Esto facilita la comu-

nicación y la cooperación entre varios departamentos, proyectos y usuarios dentro de la empresa. Los estándares se pueden definir para los nombres y los formatos de los elementos de datos, los formatos de visualización, las estructuras de los informes, la terminología, etcétera. El DBA puede implementar los estándares en un entorno de base de datos centralizado más fácilmente que en un entorno donde cada grupo de usuarios tiene el control de sus propios archivos y software.

Tiempo de desarrollo de aplicación reducido. Uno de los principales reclamos de venta de la metodología de bases de datos es que se necesita muy poco tiempo para desarrollar una aplicación nueva (como la recuperación de ciertos datos de la base de datos para imprimir un informe nuevo). El diseño y la implementación de una base de datos nueva desde el principio puede llevar más tiempo que escribir una aplicación de archivos especializada. No obstante, una vez que la base de datos está operativa y en ejecución, por lo general se necesita mucho menos tiempo para crear aplicaciones nuevas utilizando los servicios del DBMS. Se estima que el tiempo de desarrollo utilizando un DBMS es de una sexta a una cuarta parte del necesario para un sistema de archivos tradicional.

Flexibilidad. Puede ser necesario cambiar la estructura de una base de datos a medida que cambian los requisitos. Por ejemplo, puede surgir un nuevo grupo de usuarios que necesita información que actualmente no hay en la base de datos. En respuesta, puede que sea necesario añadir un archivo a la base de datos o extender los elementos de datos de un archivo existente. Los DBMS modernos permiten ciertos tipos de cambios evolutivos en la estructura de la base de datos sin que ello afecte a los datos almacenados y a los programas de aplicación existentes.

Disponibilidad de la información actualizada. Un DBMS hace que la base de datos esté disponible para todos los usuarios. Tan pronto como se aplica la actualización de un usuario a la base de datos, todos los demás usuarios pueden ver esa actualización inmediatamente. Esta disponibilidad de información actualizada es esencial para muchas de las aplicaciones de procesamiento de transacciones, como las bases de datos de los sistemas de reservas o bancarios, y esto es posible a los subsistemas de control de la concurrencia y de recuperación de un DBMS.

Economías de escala. La metodología DBMS permite la consolidación de los datos y las aplicaciones, lo que reduce el derroche de superposición entre las actividades del personal de procesamiento de datos en diferentes proyectos o departamentos, así como las redundancias entre las aplicaciones. Esto permite que toda la organización invierta en procesadores más potentes, dispositivos de almacenamiento o aparatos de comunicación, en lugar de que cada departamento compre sus propios equipos (menos potentes). De este modo se reducen los costes globales de funcionamiento y administración.

1.7 Breve historia de las aplicaciones de bases de datos

Esta sección ofrece una breve historia de las aplicaciones que utilizan DBMSs y cómo estas aplicaciones supusieron el impulso de nuevos tipos de sistemas de bases de datos.

1.7.1 Las primeras aplicaciones de bases de datos que utilizaron sistemas jerárquicos y de red

Muchas de las primeras aplicaciones de bases de datos almacenaban registros en grandes organizaciones, como corporaciones, universidades, hospitales y bancos. En muchas de esas aplicaciones había muchos registros de estructura parecida. Por ejemplo, en una aplicación para universidades, era preciso mantener información parecida por cada estudiante, cada curso y cada especialidad, etcétera. También había muchos tipos de registros y muchas interrelaciones entre ellos.

Uno de los principales problemas con los primeros sistemas de bases de datos era la mezcla de relaciones conceptuales con el almacenamiento físico y la ubicación de los registros en el disco. Por ejemplo, los registros de especialidad de un estudiante en particular podían guardarse físicamente a continuación del registro del estudiante. Aunque esto ofrecía un acceso muy eficaz para las consultas y las transacciones originales para las que fue diseñada la base de datos, no proporcionaba suficiente flexibilidad para acceder eficazmente a los registros cuando se identificaban consultas y transacciones nuevas. En particular, era muy difícil implementar con eficacia las consultas nuevas que requerían una organización diferente del almacenamiento para un procesamiento eficaz. También era muy laborioso reorganizar la base de datos cuando había cambios en los requisitos de la aplicación.

Otro defecto de los primeros sistemas era que sólo proporcionaban interfaces de lenguaje de programación. La implementación de consultas y transacciones nuevas llevaba mucho tiempo y era costosa, pues había que escribir, probar y depurar programas nuevos. La mayoría de esos sistemas de bases de datos se implantaron en grandes y costosos computadores *mainframe* a mediados de la década de 1960, y a lo largo de las décadas de 1970 y 1980. Los principales tipos de esos sistemas estaban basados en tres paradigmas principales: sistemas jerárquicos, sistemas basados en un modelo de red y sistemas de archivos inversos.

1.7.2 Flexibilidad de aplicación con las bases de datos relacionales

Las bases de datos relacionales se propusieron originalmente para separar el almacenamiento físico de los datos de su representación conceptual, así como para proporcionar una base matemática para el almacenamiento de contenidos. El modelo de datos relacional también introdujo lenguajes de consulta de alto nivel que proporcionaban una alternativa a las interfaces de lenguaje de programación; por tanto, era mucho más rápido escribir consultas nuevas. La representación relacional de los datos se parece al ejemplo presentado en la Figura 1.2. Los sistemas relacionales estaban destinados inicialmente a las mismas aplicaciones que los primitivos sistemas, pero estaban pensados para ofrecer flexibilidad en el desarrollo de nuevas consultas y para reorganizar la base de datos cuando cambiaran los requisitos.

Los sistemas relacionales experimentales desarrollados a finales de la década de 1970 y los sistemas de administración de bases de datos relacionales (RDBMS) comerciales que aparecieron a principios de la década de 1980 eran muy lentos, pues no utilizaban punteros de almacenamiento físico o la ubicación del registro para acceder a los registros de datos relacionados. Su rendimiento mejoró con el desarrollo de nuevas técnicas de almacenamiento e indexación y unas técnicas mejores de procesamiento y optimización. Eventualmente, las bases de datos relacionales se convirtieron en el tipo de sistema de bases de datos predominante para las aplicaciones de bases de datos tradicionales. En casi todos los tipos de computadores, desde los pequeños computadores personales hasta los grandes servidores, existen bases de datos relacionales.

1.7.3 Aplicaciones orientadas a objetos y la necesidad de bases de datos más complejas

El surgimiento de los lenguajes de programación orientados a objetos en la década de 1980 y la necesidad de almacenar y compartir objetos estructurados complejos induce al desarrollo de las bases de datos orientadas a objetos (OODB). Inicialmente, las OODB estaban consideradas como competidoras de las bases de datos relacionales, porque proporcionaban más estructuras de datos generales. También incorporaban muchos de los útiles paradigmas de la orientación a objetos, como los tipos de datos abstractos, la encapsulación de operaciones, la herencia y la identidad de objeto. No obstante, la complejidad del modelo y la carencia de un estándar contribuyó a su limitado uso. Ahora se utilizan principalmente en las aplicaciones especializadas (por ejemplo, en ingeniería, publicación multimedia y sistemas de fabricación). A pesar de las expectativas de que iban a provocar un gran impacto, lo cierto es que su penetración global en el mercado de productos de bases de datos permanece aún hoy por debajo del 50%.

1.7.4 Intercambio de datos en la Web para el comercio electrónico

La World Wide Web proporciona una gran red de computadores interconectados. Los usuarios pueden crear documentos utilizando un lenguaje de publicación web, como HTML (Lenguaje de marcado de hipertexto, *HyperText Markup Language*), y almacenar esos documentos en servidores web desde los que otros usuarios (clientes) pueden acceder a ellos. Los documentos se pueden enlazar mediante **hipervínculos**, que son punteros a otros documentos. En la década de 1990 apareció el comercio electrónico (*e-commerce*) como una aplicación trascendental en la Web. Cada vez iba siendo más evidente que parte de la información que aparecía en las páginas web de *e-commerce* a menudo eran datos que se extraían dinámicamente de unos DBMSs. Se desarrollaron varias técnicas que permitían el intercambio de datos en la Web. Actualmente, XML (Lenguaje de marcado extendido, *eXtended Markup Language*) está considerado como el principal estándar para el intercambio de datos entre varios tipos de bases de datos y páginas web. XML combina conceptos de los modelos utilizados en los sistemas de documentación con conceptos de modelado de bases de datos. El Capítulo 27 está dedicado a la explicación de XML.

1.7.5 Capacidades extendidas de las bases de datos para las nuevas aplicaciones

El éxito de los sistemas de bases de datos en las aplicaciones tradicionales animó a los desarrolladores de otros tipos de aplicaciones a intentar utilizarlos. Dichas aplicaciones, de las que se ofrecen unos ejemplos a continuación, utilizaban tradicionalmente sus propias estructuras de archivos y datos especializadas:

- **Aplicaciones científicas.** Almacenan grandes cantidades de datos resultado de los experimentos científicos en áreas como la física o el mapa del genoma humano.
- **Almacenamiento y recuperación de imágenes,** desde noticias escaneadas y fotografías personales, hasta imágenes de satélite o las procedentes de procedimientos médicos, como los rayos X o el MRI (procesamiento de imágenes de resonancia magnética).
- **Almacenamiento y recuperación de vídeos,** como películas, o **videoclips,** procedentes de noticias o cámaras digitales personales.
- **Aplicaciones de minado de datos,** que analizan grandes cantidades de datos buscando ocurrencias de patrones específicos o relaciones.
- **Aplicaciones espaciales,** que almacenan las ubicaciones espaciales de datos como la información meteorológica o los mapas que se utilizan en los sistemas de información geográfica.
- **Aplicaciones de series cronológicas** que almacenan información como datos económicos a intervalos regulares de tiempo (por ejemplo, gráficos de las ventas diarias o del producto nacional bruto mensual).

Es evidente que los sistemas relacionales básicos no eran muy adecuados para muchas de estas aplicaciones, normalmente por una o más de las siguientes razones:

- Se necesitaban estructuras de datos más complejas para modelar la aplicación que la simple representación relacional.
- Se necesitaron nuevos tipos de datos, **además** de los tipos numérico y de cadena de caracteres básicos.
- Para manipular los nuevos tipos de datos eran necesarias operaciones y construcciones de lenguaje de consulta nuevas.
- Se necesitaban nuevas estructuras de almacenamiento e indexación.

Esto llevó a que los desarrolladores de DBMS añadieran funcionalidad a sus sistemas. Parte de esa funcionalidad era de propósito general, como la incorporación de conceptos de las bases de datos orientadas a objetos

en los sistemas relacionales. Otra parte de esa funcionalidad era de propósito especial, en forma de módulos opcionales que se podían utilizar para aplicaciones específicas. Por ejemplo, los usuarios podrían comprar un módulo de series cronológicas para utilizarlo con su DBMS relacional para su aplicación de series cronológicas.

Actualmente, la mayoría de las organizaciones grandes utilizan distintos paquetes que funcionan en estrecha colaboración con **bases de datos *back-ends*.** Una base de datos *back-end* representa una o más bases de datos, seguramente de distintos fabricantes y diferentes modelos de datos, encaminado todo ello a almacenar los datos que esos paquetes manipulan para las transacciones, la generación de informes y dar respuesta a las consultas específicas. Uno de los sistemas que más se utiliza es **ERP (Planificación de recursos empresariales, *Enterprise Resource Planning*)**, que se utiliza para consolidar diferentes áreas funcionales dentro de una organización, como, por ejemplo, la producción, las ventas, la distribución, el marketing, las finanzas, los recursos humanos, etcétera. Otro tipo muy conocido de sistema es el software **CRM (Administración de las relaciones con el cliente, *Customer Relationship Management*)**, que abarca el procesamiento de pedidos y las funciones de marketing y soporte de clientes. Estas aplicaciones son compatibles con la Web para aquellos usuarios internos y externos a los que se dota de diferentes interfaces de portal web para interactuar con la base de datos *back-end*.

1.7.6 Bases de datos frente a recuperación de información

Tradicionalmente, la tecnología de bases de datos se aplica a los datos estructurados y formateados que se originan en las aplicaciones rutinarias gubernamentales, comerciales e industriales. Esta tecnología se utiliza mucho en la fabricación, las ventas, la banca, los seguros, las finanzas y la salud, donde los datos estructurados originan formularios como las facturas o los documentos de registro de pacientes. Ha habido un desarrollo concurrente de un campo denominado recuperación de información (IR, *information retrieval*) que tiene que ver con los libros, los manuscritos y distintos formularios de artículos basados en bibliotecas. Los datos se indexan, catalogan y anotan utilizando palabras clave. IR tiene que ver con la búsqueda de material basada en esas palabras clave, y con muchos de los problemas relacionados con el procesamiento de documentos y el procesamiento de texto de forma libre. Se ha realizado una cantidad considerable de trabajo en buscar texto basándose en palabras clave, buscar documentos y clasificarlos por su relevancia, clasificar el texto automáticamente, clasificar el texto por temas, etcétera. Con la llegada de la Web y la proliferación de las páginas HTML ejecutándose por miles de millones, es necesario aplicar muchas de las técnicas de IR para procesar los datos en la Web. Los datos de las páginas web son normalmente imágenes, texto y objetos que se activan y modifican dinámicamente. La recuperación de información en la Web es un problema nuevo que requiere la aplicación de técnicas de bases de datos e IR en variedad de nuevas combinaciones.

1.8 Cuándo no usar un DBMS

A pesar de las ventajas de usar un DBMS, hay algunas situaciones en las que su uso puede suponer unos sobrecostes innecesarios en los que no se incurriría con el procesamiento tradicional de archivos. Los sobrecostes de utilizar un DBMS se deben a lo siguiente:

- Inversión inicial muy alta en hardware, software y formación.
- La generalidad de que un DBMS ofrece definición y procesamiento de datos.
- Costes derivados de las funciones de seguridad, control de la concurrencia, recuperación e integridad.

Es posible que surjan otros problemas si los diseñadores y el DBA no diseñan correctamente la base de datos o si las aplicaciones de sistemas de bases de datos no se implantan correctamente. Por tanto, puede ser más deseable utilizar archivos normales en las siguientes circunstancias:

- Aplicaciones de bases de datos sencillas y bien definidas que no es previsible que cambien.

- Requisitos estrictos y en tiempo real para algunos programas que no podrían satisfacerse debido al sobrecoste de un DBMS.
- Inexistencia del acceso multiusuario a los datos.

Algunas industrias y aplicaciones prefieren no utilizar DBMSs de propósito general. Por ejemplo, muchas de las herramientas de diseño asistido por computador (CAD) que los ingenieros mecánicos y civiles utilizan, tienen archivos propietarios y software de administración de datos destinados a las manipulaciones internas de dibujos y objetos 3D. De forma parecida, los sistemas de comunicación y conmutación diseñados por empresas como AT&T eran manifestaciones precoces de software de bases de datos que se desarrolló para ejecutarse muy rápidamente con datos organizados jerárquicamente, al objeto de obtener un acceso rápido y el enrutamiento de llamadas. Asimismo, las implementaciones GIS a menudo implantaban sus propios esquemas de organización de datos para implementar eficazmente funciones relacionadas con el procesamiento de mapas, los contornos físicos, las líneas, los polígonos, etcétera. Los DBMSs de propósito general no son adecuados para su propósito.

1.9 Resumen

En este capítulo hemos definido una base de datos como una colección de datos relacionados, donde los *datos* son hechos grabados. Una base de datos típica representa algún aspecto del mundo real y es utilizada por uno o más grupos de usuarios con fines específicos. Un DBMS es un paquete de software generalizado destinado a implementar y mantener una base de datos computerizada. La base de datos y el software juntos forman un sistema de bases de datos. Hemos identificado algunas características que distinguen la metodología de bases de datos de las aplicaciones tradicionales de procesamiento de archivos, y hemos explicado las principales categorías de usuarios de las bases de datos, o *actores de la escena*. Además de los usuarios de las bases de datos, el personal de soporte, o *trabajadores entre bambalinas*, se pueden clasificar en varias categorías.

El capítulo también ofrece una lista de las capacidades que un software de DBMS debe ofrecer al DBA, los diseñadores y los usuarios para que les ayude en el diseño, la administración y el uso de una base de datos. Después, se ha ofrecido una perspectiva histórica de la evolución de las aplicaciones de bases de datos. Hemos apuntado al matrimonio de la tecnología de bases de datos con la tecnología de recuperación de información, que jugará un papel muy importante debido a la popularidad de la Web. Por último, hemos hablado de los sobrecostes de utilizar un DBMS y de algunas situaciones en las que no es ventajoso utilizar uno.

Preguntas de repaso

1.1. Defina los siguientes términos: datos, base de datos, DBMS, sistema de bases de datos, catálogo de la base de datos, independencia programa-datos, vista de usuario, DBA, usuario final, transacción enlatada, sistema de bases de datos deductivo, objeto persistente, metadatos y aplicación de procesamiento de transacciones.

1.2. ¿Qué cuatro tipos de acciones implican bases de datos? Explique brevemente cada uno de ellos.

1.3. Explique las principales características de la metodología de bases de datos y cómo difiere de los sistemas de archivos tradicionales.

1.4. ¿Cuáles son las responsabilidades del DBA y de los diseñadores de bases de datos?

1.5. ¿Cuáles son los diferentes tipos de bases de datos y usuarios? Explique las actividades principales de cada uno.

1.6. Explique las capacidades que un DBMS debe proporcionar.

1.7. Explique las diferencias entre los sistemas de bases de datos y los sistemas de recuperación de información.

Ejercicios

1.8. Identifique algunas operaciones de actualización y consultas informales que esperaría aplicar a la base de datos de la Figura 1.2.

1.9. ¿Cuál es la diferencia entre la redundancia controlada y la descontrolada? Ilustre su explicación con ejemplos.

1.10. Denomine todas las relaciones entre los registros de la base de datos de la Figura 1.2.

1.11. Ofrezca algunas vistas adicionales que otros grupos de usuarios podrían necesitar para la base de datos de la Figura 1.2.

1.12. Cite algunos ejemplos de restricciones de integridad que piense que podrían darse en la base de datos de la Figura 1.2.

1.13. Ofrezca ejemplos de sistemas en los que tenga sentido utilizar el procesamiento tradicional de archivos en lugar de una base de datos.

1.14. Considerando la Figura 1.2:

a. Si el nombre del departamento 'CC' (Ciencias de la Computación) cambia a 'CCIS' (Ciencias de la computación e Ingeniería de Software), y también cambia el prefijo correspondiente para el curso, identifique las columnas de la base de datos que deben actualizarse.

b. ¿Es posible reestructurar las columnas de las tablas CURSO, SECCIÓN y PRERREQUISITO para que sólo sea necesario modificar una columna?

Bibliografía seleccionada

El ejemplar de octubre de 1991 de *Communications of the ACM and Kim* (1995) incluye varios artículos que describen los DBMSs de la siguiente generación; muchas de las características de las bases de datos explicadas en el pasado están ahora disponibles comercialmente. El ejemplar de marzo de 1976 de *ACM Computing Surveys* ofrece una introducción a los sistemas de bases de datos; al lector interesado le puede proporcionar una perspectiva histórica.

CAPÍTULO 2

Conceptos y arquitectura de los sistemas de bases de datos

La arquitectura de los paquetes DBMS ha evolucionado desde los antiguos sistemas monolíticos, en los que todo el paquete de software DBMS era un sistema integrado, hasta los modernos paquetes DBMS con un diseño modular y una arquitectura de sistema cliente/servidor. Esta evolución es reflejo de las tendencias en computación, donde los grandes computadores *mainframe* centralizados se han sustituido por cientos de estaciones de trabajo distribuidas y computadores personales conectados a través de redes de comunicaciones a distintos tipos de servidores (servidores web, servidores de bases de datos, servidores de archivos, servidores de aplicaciones, etc.).

En una arquitectura DBMS cliente/servidor básica, la funcionalidad del sistema se distribuye entre dos tipos de módulos.[1] Un **módulo cliente** se diseña normalmente para que se pueda ejecutar en la estación de trabajo de un usuario o en un computador personal. Normalmente, las aplicaciones y las interfaces de usuario que acceden a las bases de datos se ejecutan en el módulo cliente. Por tanto, el módulo cliente manipula la interacción del usuario y proporciona interfaces amigables para el usuario, como formularios o GUIs basadas en menús. El otro tipo de módulo, denominado **módulo servidor**, manipula normalmente el almacenamiento de los datos, el acceso, la búsqueda y otras funciones. En la Sección 2.5 explicaremos más en detalle las arquitecturas cliente/servidor. En primer lugar, estudiaremos más conceptos básicos, que le permitirán tener un mayor conocimiento de las modernas arquitecturas de bases de datos.

En este capítulo veremos la terminología y los conceptos básicos que utilizaremos en todo el libro. La Sección 2.1 explica los modelos de datos y define los conceptos de esquema e instancia, que son fundamentales para el estudio de los sistemas de bases de datos. Después, explicaremos la arquitectura DBMS de tres esquemas y la independencia de los datos en la Sección 2.2; esto proporciona la perspectiva que tiene un usuario de lo que se supone que un DBMS debe hacer. En la Sección 2.3 se describen los tipos de interfaces y lenguajes que un DBMS normalmente proporciona. La Sección 2.4 ofrece un estudio del entorno software de un sistema de bases de datos. La Sección 2.5 ofrece una panorámica de distintos tipos de arquitecturas cliente/servidor. Por último, la Sección 2.6 ofrece una clasificación de los tipos de paquetes DBMS. La Sección 2.7 resume el capítulo.

El material de las Secciones 2.4 a 2.6 ofrece conceptos más detallados que pueden considerarse como complementarios del material de introducción básico.

---

[1] Como veremos en la Sección 2.5 existen variaciones de esta arquitectura cliente/servidor de dos capas sencillas.

2.1 Modelos de datos, esquemas e instancias

Una característica fundamental de la metodología de bases de datos es que ofrece algún nivel de abstracción de los datos. La **abstracción de datos** se refiere generalmente a la supresión de detalles de la organización y el almacenamiento de datos y a la relevancia de las características fundamentales para un conocimiento mejorado de los datos. Una de las características principales de la metodología de bases de datos es soportar la abstracción de datos para que diferentes usuarios puedan percibir esos datos con el nivel de detalle que prefieren. Un **modelo de datos** (colección de conceptos que se pueden utilizar para describir la estructura de una base de datos) proporciona los medios necesarios para conseguir esa abstracción.[2] Por *estructura de una base de datos* nos referimos a los tipos de datos, relaciones y restricciones que deben mantenerse para los datos. La mayoría de modelos de datos también incluyen un conjunto de **operaciones básicas** para especificar las recuperaciones y actualizaciones en la base de datos.

Además de las operaciones básicas proporcionadas por el modelo de datos, es cada vez más común incluir conceptos en el modelo de datos para especificar el **aspecto dinámico** o **comportamiento** de una aplicación de base de datos. Esto permite al diseñador de la base de datos especificar un conjunto de operaciones válidas definidas por el usuario que son permitidas en los objetos de la base de datos.[3] Un ejemplo de operación definida por usuario puede ser COMPUTE_GPA, que se puede aplicar al objeto ESTUDIANTE. Por el contrario, las operaciones genéricas para insertar, borrar, modificar o recuperar cualquier clase de objeto se incluyen a menudo en las *operaciones básicas del modelo de datos*. Los conceptos para especificar el comportamiento son fundamentales para los modelos de datos orientados a objetos (consulte los Capítulos 20 y 21), pero también se están incorporando en los modelos de datos más tradicionales. Por ejemplo, los modelos de objetos relacionales (consulte el Capítulo 22) extienden el modelo relacional básico para incluir dichos conceptos, además de otros. En el modelo de datos relacional hay una cláusula para adjuntar el comportamiento a las relaciones en forma de módulos almacenados persistentes, popularmente conocidos como procedimientos almacenados (consulte el Capítulo 9).

2.1.1 Categorías de modelos de datos

Se han propuesto muchos modelos de datos, que podemos clasificar conforme a los tipos de conceptos que utilizan para describir la estructura de la base de datos. Los **modelos de datos de alto nivel** o **conceptuales** ofrecen conceptos muy cercanos a como muchos usuarios perciben los datos, mientras que los **modelos de datos de bajo nivel** o **físicos** ofrecen conceptos que describen los detalles de cómo se almacenan los datos en el computador. Los conceptos ofrecidos por los modelos de datos de bajo nivel están pensados principalmente para los especialistas en computadores, no para los usuarios finales normales. Entre estos dos extremos hay una clase de **modelos de datos representativos** (o de **implementación**),[4] que ofrecen conceptos que los usuarios finales pueden entender pero que no están demasiado alejados de cómo se organizan los datos dentro del computador. Los modelos de datos representativos ocultan algunos detalles relativos al almacenamiento de los datos, pero pueden implementarse directamente en un computador.

Los modelos de datos conceptuales utilizan conceptos como entidades, atributos y relaciones. Una entidad representa un objeto o concepto del mundo real, como un empleado o un proyecto que se describe en la base de datos. Un atributo representa alguna propiedad de interés que describe a una entidad, como, por ejemplo,

---

[2] A veces, la palabra *modelo* se utiliza para denotar una descripción de base de datos específica, o esquema (por ejemplo, *el modelo de datos de marketing*). No utilizaremos esta interpretación.

[3] La inclusión de conceptos para describir el comportamiento refleja una tendencia según la cual las actividades de diseño de bases de datos y software se combinan cada vez más en una sola actividad. Tradicionalmente, la declaración de un comportamiento se asocia con el diseño de software.

[4] El término *modelo de datos de implementación* no es un término estándar; lo hemos introducido para hacer referencia a los modelos de datos disponibles en los sistemas de bases de datos comerciales.

el nombre o el salario de un empleado. Una relación entre dos o más entidades representa una asociación entre dos o más entidades; por ejemplo, una relación de trabajo entre un empleado y un proyecto. El Capítulo 3 presenta el modelo Entidad-Relación, un conocido modelo de datos conceptual de alto nivel. El Capítulo 4 describe las abstracciones adicionales que se utilizan para el modelado avanzado, como la generalización, la especialización y las categorías.

Los modelos de datos representativos o de implementación son los más utilizados en los DBMS comerciales tradicionales. Incluyen los modelos de datos relacionales ampliamente utilizados, así como los modelos de datos heredados (los modelos de red y **jerárquicos**) que tanto se han utilizado en el pasado. La Parte 2 está dedicada al modelo de datos relacional, sus operaciones y lenguajes, y algunas técnicas de programación de aplicaciones de bases de datos relacionales.[5] En los Capítulos 8 y 9 se describe el estándar SQL para las bases de datos relacionales. Los modelos de datos representativos representan los datos mediante estructuras de registro y, por tanto, se los conoce a veces como **modelos de datos basados en registros**.

Podemos considerar que el **grupo de modelos de datos de objetos (ODMG, *object data model group*)** es una nueva familia de modelos de datos de implementación de alto nivel que está más cercana a los modelos de datos conceptuales. En los Capítulos 20 y 21 se describen las características generales de las bases de datos de objetos y del estándar ODMG propuesto. Los modelos de datos de objetos también se utilizan a menudo como modelos conceptuales de alto nivel, generalmente en el ámbito de la ingeniería de software.

Los modelos de datos físicos describen cómo se almacenan los datos en el computador en forma de archivos, representando la información como formatos de registro, ordenación de registros y rutas de acceso. Una ruta de acceso es una estructura que hace más eficaz la búsqueda de registros en una base de datos. En los Capítulos 13 y 14 explicaremos las técnicas de almacenamiento físico y las estructuras de acceso. Un **índice** es un ejemplo de ruta de acceso que permite el acceso directo a los datos que utilizan un término del índice o una palabra clave. Es parecido al índice final de este libro, excepto que se puede organizar lineal o jerárquicamente, o de algún otro modo.

2.1.2 Esquemas, instancias y estado de la base de datos

En cualquier modelo de datos es importante distinguir entre la *descripción* de la base de datos y la *misma base de datos*. La descripción de una base de datos se denomina **esquema de la base de datos**, que se especifica durante la fase de diseño y no se espera que cambie con frecuencia.[6] La mayoría de los modelos de datos tienen ciertas convenciones para la visualización de los esquemas a modo de diagramas. Un esquema visualizado se denomina **diagrama del esquema**. La Figura 2.1 muestra un diagrama del esquema para la base de datos de la Figura 1.2; el diagrama muestra la estructura de cada tipo de registro, pero no las instancias reales de los registros. A cada objeto del esquema (como ESTUDIANTE o CURSO) lo denominamos **estructura de esquema**.

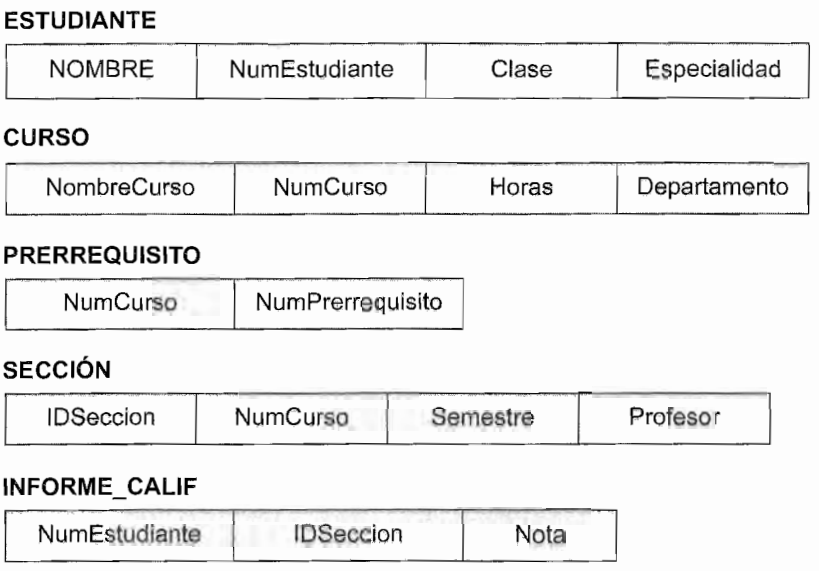
Un diagrama del esquema sólo muestra *algunos aspectos* de un esquema, como los nombres de los tipos de registros y los elementos de datos, y algunos tipos de restricciones. Otros aspectos no se especifican; por ejemplo, la Figura 2.1 no muestra los tipos de datos de cada elemento de datos, ni las relaciones entre los distintos archivos. En los diagramas de esquemas no se representan muchos de los tipos de restricciones. Una restricción como, por ejemplo, "*los estudiantes que se especializan en ciencias de la computación deben terminar CC1310 antes de finalizar su curso de segundo año*", es muy difícil de representar.

Los datos reales de una base de datos pueden cambiar con mucha frecuencia. Por ejemplo, la base de datos de la Figura 1.2 cambia cada vez que se añade un estudiante o se introduce una calificación nueva. Los

[5] En los Apéndices E y F se incluye un resumen de los modelos de datos de red y jerárquicos. Son accesibles desde el sitio web *www.librosite.net/elmasri.*

[6] Normalmente, es necesario hacer cambios en el esquema cuando cambian los requisitos de las aplicaciones de bases de datos. Los sistemas de bases de datos más nuevos incluyen operaciones para permitir cambios en el esquema, aunque el proceso de cambio del esquema es más complejo que las sencillas actualizaciones de la base de datos.

Figura 2.1. Diagrama del esquema para la base de datos de la Figura 1.2.



datos de la base de datos en un momento concreto se denominan **estado de la base de datos** o ***snapshot* (captura)**. También reciben el nombre de conjunto *actual* de **ocurrencias o instancias** de la base de datos. En un estado dado de la base de datos, cada estructura de esquema tiene su propio *conjunto actual* de instancias; por ejemplo, la construcción ESTUDIANTE contendrá el conjunto de entidades estudiante individuales (registros) como sus instancias. Es posible construir muchos estados de la base de datos para que se correspondan con un esquema de bases de datos particular. Cada vez que se inserta o borra un registro, o cambia el valor de un elemento de datos de un registro, cambia el estado de la base de datos por otro.

Esta distinción entre esquema de la base de datos y estado de la base de datos es muy importante. Cuando **definimos** una base de datos nueva, sólo especificamos su esquema al DBMS. A estas alturas, el estado correspondiente de la base de datos es el *estado vacío*, sin datos. El *estado inicial* de la base de datos se da cuando ésta se **rellena o carga** por primera vez con los datos iniciales. Desde ese momento, cada vez que sobre la base de datos se aplica una operación de actualización, obtenemos otro estado de la base de datos. En cualquier momento en el tiempo, la base de datos tiene un *estado actual*.[7] El DBMS es en parte responsable de garantizar que cada estado de la base de datos sea un **estado válido**; es decir, un estado que satisfaga la estructura y las restricciones especificadas en el esquema. Por tanto, especificar un esquema correcto al DBMS es sumamente importante, por lo que el esquema debe diseñarse con sumo cuidado. El DBMS almacena las descripciones de las construcciones de esquema y las restricciones (también denominadas **metadatos**) en el catálogo del DBMS, para que el software DBMS pueda dirigirse al esquema siempre que lo necesite. En ocasiones, el esquema recibe el nombre de **intención**, y el estado de la base de datos **extensión** del esquema.

Aunque, como ya mencionamos, no se supone que el esquema cambie con frecuencia, no es raro que ocasionalmente haya que introducir algún cambio en él al cambiar los requisitos de la aplicación. Por ejemplo, podemos decidir que es necesario almacenar otro elemento de datos por cada registro del archivo, como añadir Fecha_nac al esquema ESTUDIANTE de la Figura 2.1. Esto se conoce como evolución del esquema. Los DBMS más modernos incluyen algunas operaciones para la **evolución del esquema** que se pueden aplicar mientras la base de datos es operativa.

[7] El estado actual también se denomina *snapshot actual* de la base de datos.

2.2 Arquitectura de tres esquemas e independencia de los datos

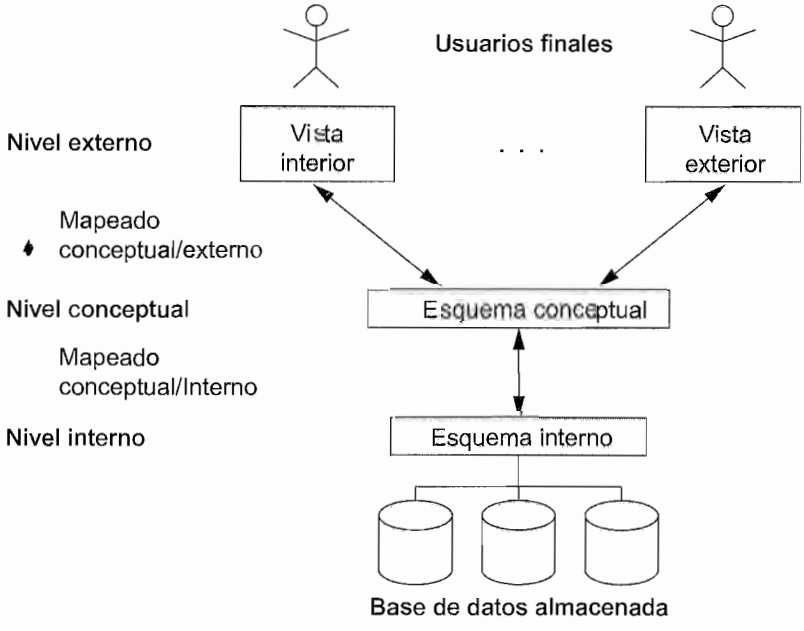
Tres de las cuatro importantes características de la metodología de bases de datos que se mencionaron en la Sección 1.3 son: (1) aislamiento de los programas y los datos (independencia programa-datos y programa-operación), (2) soporte de varias vistas de usuario y (3) uso de un catálogo para almacenar la descripción de la base de datos (esquema). En esta sección especificamos una arquitectura para los sistemas de bases de datos, denominada **arquitectura de tres esquemas**,[8] que se propuso para ayudar a conseguir y visualizar estas características. Después explicaremos el concepto de independencia de los datos.

2.2.1 Arquitectura de tres esquemas

El objetivo de la arquitectura de tres esquemas, ilustrada en la Figura 2.2, es separar las aplicaciones de usuario y las bases de datos físicas. En esta arquitectura se pueden definir esquemas en los siguientes tres niveles:

1. El **nivel interno** tiene un **esquema interno**, que describe la estructura de almacenamiento físico de la base de datos. El esquema interno utiliza un modelo de datos físico y describe todos los detalles del almacenamiento de datos y las rutas de acceso a la base de datos.
2. El **nivel conceptual** tiene un **esquema conceptual**, que describe la estructura de toda la base de datos para una comunidad de usuarios. El esquema conceptual oculta los detalles de las estructuras de almacenamiento físico y se concentra en describir las entidades, los tipos de datos, las relaciones, las operaciones de los usuarios y las restricciones. Normalmente, el esquema conceptual se describe con un modelo de datos representativo cuando se implementa un sistema de bases de datos. Este *esquema conceptual de implementación* se basa a menudo en un *diseño de esquema conceptual* en un modelo de datos de alto nivel.

Figura 2.2. Arquitectura de tres esquemas.



[8] También se conoce como arquitectura ANSI/SPARC, según el comité que la propuso (Tsichritzis and Klug 1978).

3. El **nivel de vista** o **externo** incluye una cierta cantidad de **esquemas externos** o **vistas de usuario**. Un esquema externo describe la parte de la base de datos en la que un grupo de usuarios en particular está interesado y le oculta el resto de la base de datos. Como en el caso anterior, cada esquema externo se implementa normalmente mediante un modelo de datos representativo, posiblemente basado en un diseño de esquema externo de un modelo de datos de alto nivel.

La arquitectura de tres esquemas es una buena herramienta con la que el usuario puede visualizar los niveles del esquema de un sistema de bases de datos. La mayoría de los DBMSs no separan completa y explícitamente los tres niveles, pero soportan esta arquitectura en cierta medida. Algunos DBMSs pueden incluir en el esquema conceptual detalles a nivel físico. La arquitectura de tres niveles ANSI ocupa un lugar importante en el desarrollo de tecnologías de bases de datos porque separa el nivel externo de los usuarios, el nivel conceptual del sistema y el nivel de almacenamiento interno para diseñar una base de datos. Incluso hoy en día se aplica mucho al diseño de DBMSs. En la mayoría de los DBMSs que soportan vistas de usuario, los esquemas externos se especifican en el mismo modelo de datos que describe la información a nivel conceptual (por ejemplo, un DBMS relacional como Oracle utiliza SQL para esto). Algunos DBMSs permiten el uso de diferentes modelos de datos en los niveles conceptual y externo. Un ejemplo es Base de datos universal (UDB, *Universal Data Base*), un DBMS de IBM que utiliza el modelo relacional para describir el esquema conceptual, pero puede utilizar un modelo orientado a objetos para describir un esquema externo.

Observe que los tres esquemas sólo son *descripciones* de datos; los datos almacenados que existen en realidad están en el nivel físico. En un DBMS basado en la arquitectura de tres esquemas, cada grupo de usuarios sólo se refiere a su propio esquema externo. Por tanto, el DBMS debe transformar una solicitud especificada en un esquema externo en una solicitud contra el esquema conceptual, y después en una solicitud en el esquema interno para el procesamiento sobre la base de datos almacenada. Si la solicitud es para una recuperación de la base de datos, es preciso reformatear los datos extraídos de la base de datos almacenada para que concuerden o encajen en la vista externa del usuario. Los procesos para transformar solicitudes y resultados entre niveles se denominan mapeados. Estos mapeados pueden consumir bastante tiempo, por lo que algunos DBMS (sobre todo los que están pensados para bases de datos pequeñas) no soportan las vistas externas. No obstante, incluso en dichos sistemas se necesita algo de mapeado para transformar las solicitudes entre los niveles conceptual e interno.

2.2.2 Independencia de los datos

La arquitectura de tres esquemas se puede utilizar para explicar el concepto de **independencia de los datos**, que puede definirse como la capacidad de cambiar el esquema en un nivel de un sistema de bases de datos sin tener que cambiar el esquema en el siguiente nivel más alto. Se pueden definir dos tipos de independencia de datos:

1. **Independencia lógica de datos.** Es la capacidad de cambiar el esquema conceptual sin tener que cambiar los esquemas externos o los programas de aplicación. Es posible cambiar el esquema conceptual para expandir la base de datos (añadiendo un tipo de registro o un elemento de datos), para cambiar las restricciones o para reducir la base de datos (eliminando un tipo de registro o un elemento de datos). En el último caso, no deben verse afectados los esquemas externos que sólo se refieren a los datos restantes. Por ejemplo, el esquema externo de la Figura 1.5(a) no debe verse afectado por cambiar el archivo INFORME_CALIF (o tipo de registro) de la Figura 1.2 por el mostrado en la Figura 1.6(a). Sólo es necesario cambiar la definición de la vista y los mapeados en un DBMS que soporta la independencia lógica de datos. Una vez que el esquema conceptual sufre una reorganización lógica, los programas de aplicación que hacen referencia a las estructuras de esquema externo deben funcionar como antes. En el esquema conceptual se pueden introducir cambios en las restricciones sin que se vean afectados los esquemas externos o los programas de aplicación.
2. **Independencia física de datos.** Es la capacidad de cambiar el esquema interno sin que haya que cambiar el esquema conceptual. Por tanto, tampoco es necesario cambiar los esquemas externos.

Puede que haya que realizar cambios en el esquema interno porque algunos archivos físicos fueran reorganizados (por ejemplo, por la creación de estructuras de acceso adicionales) de cara a mejorar el rendimiento de las recuperaciones o las actualizaciones. Si en la base de datos permanecen los mismos datos que antes, no hay necesidad de cambiar el esquema conceptual. Por ejemplo, el suministro de una ruta de acceso para mejorar la velocidad de recuperación de los registros de sección (véase la Figura 1.2) por semestre y año no debe requerir modificar una consulta del tipo "*listar todas las secciones ofrecidas en otoño de 2004*", aunque el DBMS ejecutará la consulta con más eficacia utilizando la ruta de acceso nueva.

Por regla general, la independencia física de datos existe en la mayoría de las bases de datos y de los entornos de archivos en los que al usuario se le ocultan la ubicación exacta de los datos en el disco, los detalles hardware de la codificación del almacenamiento, la colocación, la compresión, la división, la fusión de registros, etcétera. Las aplicaciones siguen obviando estos detalles. Por el contrario, la independencia lógica de datos es muy difícil de conseguir porque permite los cambios estructurales y restrictivos sin afectar a los programas de aplicación (un requisito mucho más estricto).

Siempre que tengamos un DBMS de varios niveles, su catálogo debe ampliarse para incluir información de cómo mapear las consultas y los datos entre los diferentes niveles. El DBMS utiliza software adicional para acometer estos mapeados refiriéndose a la información de mapeado que hay en el catálogo. La independencia de datos ocurre porque cuando el esquema cambia a algún nivel, el esquema en el siguiente nivel más alto permanece inalterado; sólo cambia el *mapeado* entre los dos niveles. Por tanto, las aplicaciones que hacen referencia al esquema del nivel más alto no tienen que cambiar.

La arquitectura de tres esquemas puede facilitar el conseguir la independencia de datos verdadera, tanto física como lógica. Sin embargo, los dos niveles de mapeados crean un sobrecoste durante la compilación o la ejecución de una consulta o un programa, induciendo a deficiencias en el DBMS. Debido a esto, pocos DBMSs han implementado la arquitectura de tres esquemas completa.

2.3 Lenguajes e interfaces de bases de datos

En la Sección 1.4 explicamos la variedad de usuarios que un DBMS soporta. El DBMS debe proporcionar los lenguajes e interfaces apropiados para cada categoría de usuarios. En esta sección explicamos los tipos de lenguajes e interfaces proporcionados por un DBMS y las categorías de usuarios a las que se dirige cada interfaz.

2.3.1 Lenguajes DBMS

Una vez completado el diseño de una base de datos y elegido un DBMS para implementarla, el primer paso es especificar los esquemas conceptual e interno para la base de datos y cualesquiera mapeados entre los dos. En muchos DBMSs donde no se mantiene una separación estricta de niveles, el DBA y los diseñadores de la base de datos utilizan un lenguaje, denominado **lenguaje de definición de datos (DDL, *data definition language*)**, para definir los dos esquemas. El DBMS tendrá un compilador DDL cuya función es procesar las sentencias DDL a fin de identificar las descripciones de las estructuras del esquema y almacenar la descripción del mismo en el catálogo del DBMS.

En los DBMSs donde hay una clara separación entre los niveles conceptual e interno, se utiliza DDL sólo para especificar el esquema conceptual. Para especificar el esquema interno se utiliza otro lenguaje, el **lenguaje de definición de almacenamiento (SDL, *storage definition language*)**. Los mapeados entre los dos esquemas se pueden especificar en cualquiera de estos lenguajes. En la mayoría de los DBMSs relacionales actuales, no hay un lenguaje específico que asuma el papel de SDL. En cambio, el esquema interno se especifica mediante una combinación de parámetros y especificaciones relacionadas con el almacenamiento: el personal del DBA normalmente controla la indexación y la asignación de datos al almacenamiento. Para conseguir una

arquitectura de tres esquemas real se necesita un tercer lenguaje, el **lenguaje de definición de vistas (VDL, *view definition language*)**, a fin de especificar las vistas de usuario y sus mapeados al esquema conceptual, pero en la mayoría de los DBMSs se utiliza el DDL para definir tanto el esquema conceptual como el externo. En los DBMSs relacionales se utiliza SQL actuando como VDL para definir las vistas de usuario o de aplicación como resultado de las consultas predefinidas (consulte los Capítulos 8 y 9).

Una vez compilados los esquemas de la base de datos y rellenada ésta con datos, los usuarios deben disponer de algunos medios para manipularla. Entre las manipulaciones típicas podemos citar la recuperación, la inserción, el borrado y la modificación de datos. El DBMS proporciona un conjunto de operaciones o un lenguaje denominado **lenguaje de manipulación de datos (DML, *data manipulation language*)** para todas estas tareas.

En los DBMSs actuales, los tipos de lenguajes anteriormente citados normalmente *no están considerados como lenguajes distintos*; más bien, se utiliza un lenguaje integrado comprensivo que incluye construcciones para la definición del esquema conceptual, la definición de vistas y la manipulación de datos. La definición del almacenamiento normalmente se guarda aparte, ya que se utiliza para definir las estructuras de almacenamiento físico a fin de refinar el rendimiento del sistema de bases de datos, que normalmente lo lleva a cabo el personal del DBA. El lenguaje de bases de datos relacionales SQL es un ejemplo típico de lenguaje de bases de datos comprensible (consulte los Capítulos 8 y 9). SQL representa una combinación de DDL, VDL y DML, así como sentencias para la especificación de restricciones, la evolución del esquema y otras características. El SDL era un componente de las primeras versiones de SQL, pero se ha eliminado del lenguaje para mantenerlo únicamente en los niveles conceptual y externo.

Hay dos tipos principales de DML. Se puede utilizar un DML de **alto nivel** o **no procedimental** para especificar de forma concisa las operaciones complejas con las bases de datos. Muchos DBMS admiten sentencias DML de alto nivel mediante la introducción interactiva desde el monitor o terminal, o incrustadas en un lenguaje de programación de propósito general. En el último caso, las sentencias DML deben identificarse dentro del programa para que el precompilador las pueda extraer y el DBMS las pueda procesar. Un DML de **bajo nivel** o **procedimental** *debe* incrustarse en un lenguaje de programación de propósito general. Normalmente, este tipo de DML recupera registros individuales u objetos de la base de datos, y los procesa por separado. Por consiguiente, es preciso utilizar construcciones de un lenguaje de programación, como los bucles, para recuperar y procesar cada registro de un conjunto de registros. Los DMLs de bajo nivel también se conocen con el nombre de DMLs ***record-at-a-time*** (registro de una sola vez), debido a esta propiedad. DL/1, un DML diseñado para el modelo jerárquico, es un DML de bajo nivel que utiliza comandos como GET UNIQUE, GET NEXT o GET NEXT WITHIN PARENT para navegar de un registro a otro dentro de la jerarquía de registros de una base de datos. Los DMLs de alto nivel, como SQL, pueden especificar y recuperar muchos registros con una sola sentencia DML; por tanto, también se conocen como DML ***set-at-a-time*** o ***set-oriented*** (un conjunto de una sola vez, u orientado a conjuntos). Una consulta en un DML de alto nivel a menudo especifica los datos que hay que recuperar, en lugar de cómo recuperarlos; en consecuencia, dichos lenguajes también se conocen como **declarativos**.

Siempre que hay comandos DML, de alto o de bajo nivel, incrustados en un lenguaje de programación de propósito general, ese lenguaje se denomina **lenguaje *host***, y el DML **sublenguaje de datos**.[9] Por el contrario, un DML de alto nivel utilizado de forma interactiva independiente se conoce como **lenguaje de consulta**. En general, tanto los comandos de recuperación como los de actualización de un DML de alto nivel se pueden utilizar interactivamente y, por tanto, se consideran como parte del lenguaje de consulta.[10]

---

[9] En las bases de datos de objetos, los sublenguajes de *host* y datos normalmente forman un lenguaje integrado (por ejemplo, C++ con algunas extensiones a fin de soportar la funcionalidad de bases de datos). Algunos sistemas relacionales también proporcionan lenguajes integrados (por ejemplo, PL/SQL de Oracle).

[10] Según el significado en inglés de la palabra "*query*" (consulta), realmente se debería utilizar para describir sólo las recuperaciones, no las actualizaciones.

Los usuarios finales casuales normalmente utilizan un lenguaje de consulta de alto nivel para especificar sus consultas, mientras que los programadores utilizan el DML en su forma incrustada. Los usuarios principiantes y paramétricos normalmente utilizan **interfaces amigables para el usuario** para interactuar con la base de datos; los usuarios casuales, u otros usuarios, que quieren aprender los detalles de un lenguaje de consulta de alto nivel también pueden utilizar estas interfaces. A continuación explicamos los tipos de interfaces.

2.3.2 Interfaces de los DBMSs

El DBMS puede incluir las siguientes interfaces amigables para el usuario:

Interfaces basadas en menús para los clientes web o la exploración. Estas interfaces presentan al usuario listas de opciones (denominadas **menús**) que le guían por la formulación de una consulta. Los menús eliminan la necesidad de memorizar los comandos específicos y la sintaxis de un lenguaje de consulta; en cambio, la consulta se compone paso a paso eligiendo opciones de los menús visualizados por el sistema. Los menús desplegables son una técnica muy popular en las **interfaces de usuario basadas en la Web**. También se utilizan a menudo en las **interfaces de exploración**, que permiten al usuario examinar los contenidos de una base de datos de forma indagatoria y desestructurada.

Interfaces basadas en formularios. Una interfaz basada en formularios muestra un formulario a cada usuario. Los usuarios pueden rellenar las entradas del **formulario** para insertar datos nuevos, o rellenar únicamente ciertas entradas, en cuyo caso el DBMS recuperará los datos coincidentes para el resto de entradas. Normalmente, los formularios se diseñan y programan para los usuarios principiantes como interfaces para las transacciones enlatadas. Muchos DBMSs tienen **lenguajes de especificación de formularios**, que son lenguajes especiales que ayudan a los programadores a especificar dichos formularios. SQL*Forms es un lenguaje basado en formularios que especifica consultas utilizando un formulario diseñado en combinación con el esquema de la base de datos relacional. Oracle Forms es un componente de la suite de productos de Oracle que proporciona un amplio conjunto de características para diseñar y construir aplicaciones mediante formularios. Algunos sistemas tienen utilidades que definen un formulario dejando que el usuario final construya interactivamente en pantalla un formulario de ejemplo.

Interfaces gráficas de usuario. Una GUI normalmente muestra un esquema al usuario de forma esquemática. El usuario puede especificar entonces una consulta manipulando el diagrama. En muchos casos, las GUIs utilizan tanto menús como formularios. La mayoría de las GUIs utilizan un **dispositivo apuntador**, como el ratón, para elegir distintas partes del diagrama esquemático visualizado.

Interfaces de lenguaje natural. Estas interfaces aceptan consultas escritas en inglés u otro idioma e intentan *entenderlas*. Una interfaz de este tipo normalmente tiene su propio *esquema*, que es parecido al esquema conceptual de la base de datos, así como un diccionario de palabras importantes. La interfaz de lenguaje natural se refiere a las palabras de su esquema, así como al conjunto de palabras estándar de su diccionario, para interpretar la consulta. Si la interpretación es satisfactoria, la interfaz genera una consulta de alto nivel correspondiente a la consulta de lenguaje natural y la envía al DBMS para que la procese; de lo contrario, se inicia un diálogo con el usuario para clarificar la consulta. Las capacidades de las interfaces de lenguaje natural no avanzan rápidamente. En la actualidad vemos motores de búsqueda que aceptan cadenas de palabras en lenguajes naturales (por ejemplo, inglés o español) y las intentan emparejar con documentos de sitios específicos (en el caso de motores de búsqueda locales) o con páginas de la Web (para motores como Google o AskJeeves). Utilizan índices de palabras predefinidos y funciones de clasificación para recuperar y presentar los documentos resultantes según un grado decreciente de coincidencia. Estas interfaces de consulta textual de "formato libre" no son muy comunes en las bases de datos estructuradas relacionales o heredadas.

Entrada y salida de lenguaje hablado. Cada vez es más común el uso limitado del habla como consulta de entrada y como respuesta a una pregunta o como resultado de una consulta. Las aplicaciones con vocabularios limitados como las búsquedas en los directorios telefónicos, la salida/llegada de vuelos y la informa-

ción sobre el estado de las cuentas bancarias permiten la entrada y salida en lenguaje hablado para que las personas tengan acceso a esta información. La entrada de lenguaje hablado se detecta mediante una librería de palabras predefinidas que se utilizan para configurar los parámetros que se suministran a las consultas. Para la salida, se realiza una conversión parecida del texto o los números al lenguaje hablado.

Interfaces para los usuarios paramétricos. Los usuarios paramétricos, como los cajeros automáticos, a menudo tienen un pequeño conjunto de operaciones que se deben llevar a cabo repetidamente. Por ejemplo, un cajero puede utilizar teclas de función para invocar transacciones rutinarias y repetitivas, como depósitos o retiradas de las cuentas, o consultas de saldo. Los analistas de sistemas y los programadores diseñan e implementan una interfaz especial por cada clase de usuarios principiantes. Normalmente, se incluye un pequeño conjunto de comandos abreviados, con el objetivo de minimizar el número de pulsaciones necesarias por cada consulta. Por ejemplo, las teclas de función de un terminal se pueden programar para iniciar determinados comandos. Esto permite que el usuario paramétrico proceda con una cantidad mínima de pulsaciones.

Interfaces para el DBA. La mayoría de los sistemas de bases de datos contienen comandos privilegiados que sólo puede utilizar el personal del DBA. Entre ellos hay comandos para crear cuentas, configurar los parámetros del sistema, conceder la autorización de una cuenta, cambiar un esquema y reorganizar las estructuras de almacenamiento de una base de datos.

2.4 Entorno de un sistema de bases de datos

Un DBMS es un sistema de software complejo. En esta sección explicamos los tipos de componentes software que constituyen un DBMS y los tipos de software de computador con el que el DBMS interactúa.

2.4.1 Módulos componentes de un DBMS

La Figura 2.3 ilustra, de una forma sencilla, los componentes típicos de un DBMS. La figura está dividida en dos niveles: la mitad superior se refiere a los diversos usuarios del entorno de base de datos y sus interfaces; la mitad inferior muestra las "entrañas" del DBMS responsables del almacenamiento de datos y el procesamiento de transacciones.

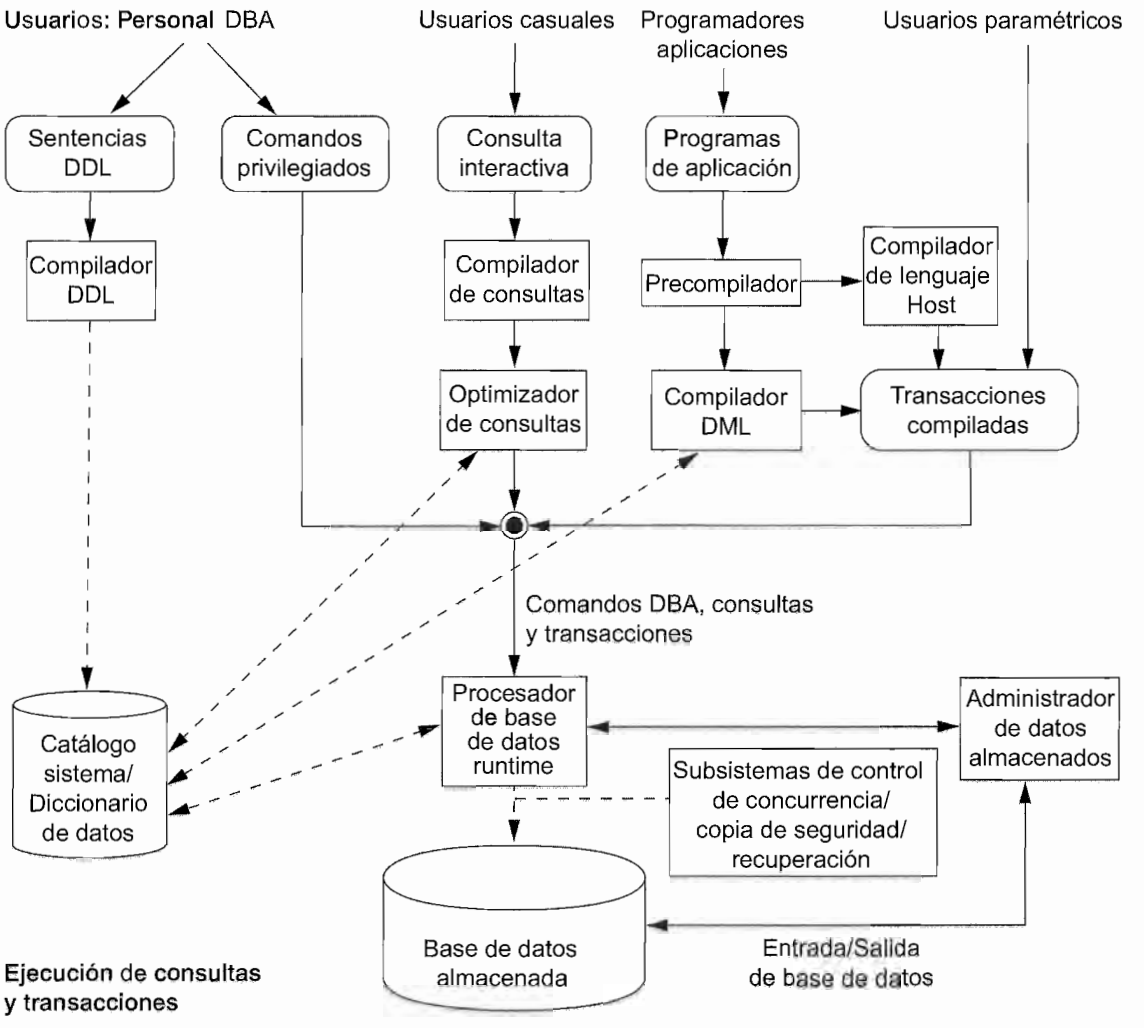
La base de datos y el catálogo del DBMS normalmente se almacenan en el disco. El acceso al disco está principalmente controlado por el **sistema operativo (SO)**, que planifica la entrada/salida del disco. Un módulo **administrador de los datos almacenados** de alto nivel del DBMS controla el acceso a la información del DBMS almacenada en el disco, sea parte de la base de datos o del catálogo.

Vamos a ver primero la parte superior de la figura. Muestra las interfaces para el personal del DBA, los usuarios casuales que trabajan con interfaces interactivas para formular consultas, los programadores de aplicaciones que programan utilizando algunos lenguajes *host* y los usuarios paramétricos que realizan las entradas de datos suministrando parámetros a las transacciones predefinidas. El personal del DBA trabaja en definir la base de datos y refinarla introduciendo cambios en su definición mediante el DDL y otros comandos privilegiados.

El compilador DDL procesa las definiciones de esquema, especificadas en el DDL, y almacena las descripciones de los esquemas (metadatos) en el catálogo del DBMS. El catálogo incluye información como los nombres y los tamaños de los archivos, los nombres y los tipos de datos de los elementos de datos, los detalles del almacenamiento de cada archivo, la información de mapeado entre esquemas y las restricciones, además de muchos otros tipos de información que los módulos del DBMS necesitan. Los módulos software del DBMS buscan después en el catálogo la información que necesitan.

Los usuarios casuales y las personas con una necesidad ocasional de información de la base de datos interactúan utilizando alguna forma de interfaz, que mostramos, como la **interfaz de consulta interactiva**. No mostramos explícitamente ninguna interacción basada en menús o en formularios que puede utilizarse para

Figura 2.3. Módulos constituyentes de un DBMS y sus interacciones.



generar automáticamente la consulta interactiva. Un **compilador de consultas** analiza estas consultas sintácticamente para garantizar la corrección de las operaciones de los modelos, los nombres de los elementos de datos, etcétera, y luego lo compila todo en un formato interno. Esta consulta interna está sujeta a la optimización de la consulta que explicaremos en el Capítulo 15. Entre otras cosas, el **optimizador de consultas** se ocupa de la reconfiguración y la posible reordenación de operaciones, eliminación de redundancias y uso de los algoritmos e índices correctos durante la ejecución. Consulta el catálogo del sistema para información estadística y física acerca de los datos almacenados, y genera un código ejecutable que lleva a cabo las operaciones necesarias para la consulta y realiza las llamadas al procesador *runtime*.

Los programadores de aplicaciones escriben programas en lenguajes *host* como Java, C o COBOL, que son enviados a un **precompilador**. Éste extrae los comandos DML de un programa de aplicación escrito en un lenguaje de programación *host*.

Estos comandos se envían al compilador DML para su compilación en código objeto y así poder acceder a la base de datos. El resto del programa se envía al compilador de lenguaje *host*. Los códigos objeto para los

comandos DML y el resto del programa se enlazan, formando una transacción enlatada cuyo código ejecutable incluye llamadas al procesador de base de datos *runtime*. Estas transacciones enlatadas resultan de utilidad para los usuarios paramétricos que simplemente suministran los parámetros a dichas transacciones, de modo que pueden ejecutarse repetidamente como transacciones separadas. Un ejemplo es una transacción de retirada de fondos donde el número de cuenta y la cantidad pueden suministrarse como parámetros.

En la parte inferior de la Figura 2.3 aparece el procesador de bases de datos *runtime* para ejecutar (1) los comandos privilegiados, (2) los proyectos de consultas ejecutables y (3) las transacciones enlatadas con parámetros *runtime*. Trabaja con el diccionario del sistema y se puede actualizar con estadísticas. Funciona con el administrador de datos almacenados, que a su vez utiliza servicios básicos del sistema operativo para ejecutar operaciones de entrada/salida de bajo nivel entre el disco y la memoria principal. Se encarga de otros aspectos de la transferencia de datos como la administración de los búferes en la memoria principal. Algunos DBMSs tienen su propio módulo de administración de búfer, mientras que otros dependen del SO para dicha administración. En esta figura hemos puesto por separado el control de la concurrencia y los sistemas de copia de seguridad y recuperación como un módulo. Con propósitos de administración de transacciones, están integrados en el funcionamiento del procesador de bases de datos *runtime*.

Ahora es normal tener el **programa cliente** que accede al DBMS ejecutándose en un computador diferente al que alberga la base de datos. El primero se denomina **computador cliente** y ejecuta un cliente DBMS, y el último se denomina **servidor de bases de datos**. En algunos casos, el cliente accede a un computador intermedio, denominado **servidor de aplicaciones**, que a su vez accede al servidor de bases de datos. Elaboraremos este tema en la Sección 2.5.

La Figura 2.3 no tiene la intención de describir un DBMS específico; en cambio, ilustra los módulos DBMS típicos. El DBMS interactúa con el sistema operativo cuando se necesita acceso al disco (a la base de datos o al catálogo). Si varios usuarios comparten el computador, el SO planificará las peticiones de acceso al disco DBMS y el procesamiento DBMS junto con otros procesos. Por el contrario, si el computador está principalmente dedicado a ejecutar el servidor de bases de datos, el DBMS controlará el *buffering* de memoria principal de las páginas de disco. El DBMS también interactúa con los compiladores en el caso de lenguajes de programación *host* de propósito general, y con los servidores de aplicaciones y los programas cliente que se ejecutan en máquinas separadas a través de la interfaz de red del sistema.

2.4.2 Utilidades del sistema de bases de datos

Además de poseer los módulos software recientemente descritos, la mayoría de los DBMSs tienen utilidades de bases de datos que ayudan al DBA a administrar el sistema de bases de datos. Las funciones de las utilidades más comunes son las siguientes:

- **Carga**. La carga de los archivos de datos existentes (como archivos de texto o archivos secuenciales) en la base de datos se realiza con una utilidad de carga. Normalmente, a la utilidad se le especifican el formato (origen) actual del archivo de datos y la estructura del archivo de base de datos (destino) deseada; después, reformatea automáticamente los datos y los almacena en la base de datos. Con la proliferación de DBMSs, la transferencia de datos de un DBMS a otro es cada vez más común en muchas empresas. Algunos fabricantes están ofreciendo productos que generan los programas de carga apropiados, dando las descripciones de almacenamiento de base de datos de origen y de destino existentes (esquemas internos). Estas herramientas también se conocen como **herramientas de conversión**. Para el DBMS jerárquico denominado IMS (IBM) y para muchos DBMSs de red como IDMS (Computer Associates), SUPRA (Cincom) o IMAGE (HP), los fabricantes o terceros están desarrollando toda una variedad de herramientas de conversión (por ejemplo, SUPRA Server SQL de Cincom) para transformar los datos en el modelo relacional.
- **Copia de seguridad**. Una utilidad de copia de seguridad crea una copia de respaldo de la base de datos, normalmente descargando la base de datos entera en una cinta. La copia de seguridad se puede utilizar

para restaurar la base de datos en caso de un fallo desastroso. También se suelen utilizar las copias de seguridad incrementales, con las que sólo se hace copia de los cambios experimentados por la base de datos desde la última copia. La copia de seguridad incremental es más compleja, pero ahorra espacio.

- **Reorganización del almacenamiento de la base de datos**. Esta utilidad se puede utilizar para reorganizar un conjunto de archivos de bases de datos en una organización de archivos diferente a fin de mejorar el rendimiento.
- **Monitorización del rendimiento**. Una utilidad de este tipo monitoriza el uso de la base de datos y ofrece estadísticas al DBA. Este último utiliza las estadísticas para tomar decisiones, como si debe o no reorganizar los archivos, o si tiene que añadir o eliminar índices para mejorar el rendimiento.

Hay otras utilidades para ordenar archivos, manipular la compresión de datos, monitorizar el acceso de los usuarios, interactuar con la red y llevar a cabo otras funciones.

2.4.3 Herramientas, entornos de aplicación e instalaciones de comunicaciones

También existen otras herramientas para los diseñadores de bases de datos, usuarios y DBMS. Las herramientas CASE[11] se utilizan en la fase de diseño de los sistemas de bases de datos. Otra herramienta que puede resultar muy útil en empresas grandes es un **sistema de diccionario de datos** (o **almacén de datos**) ampliado. Además de almacenar información de catálogo sobre esquemas y restricciones, el diccionario de datos almacena otra información, como decisiones de diseño, uso de estándares, descripciones de las aplicaciones e información de usuario. Dicho sistema también se denomina **almacén de información**. Los usuarios o el DBA pueden acceder *directamente* a esta información siempre que lo necesiten. Una utilidad de diccionario de datos es parecida al catálogo del DBMS, pero incluye una amplia variedad de información a la que acceden principalmente los usuarios, más que el software de DBMS.

Los **entornos de desarrollo de aplicaciones**, como PowerBuilder (Sybase) o JBuilder (Borland), son cada vez más populares. Estos sistemas proporcionan un entorno para el desarrollo de aplicaciones de bases de datos e incluyen servicios que ayudan en muchos de los aspectos de los sistemas de bases de datos, como el diseño de la base de datos, el desarrollo de la GUI, las consultas y las actualizaciones, y el desarrollo de una aplicación.

El DBMS también necesita interactuar con **software de comunicaciones**, cuya función es permitir a los usuarios de ubicaciones alejadas (remotas) del sistema de bases de datos acceder a la base de datos a través de sus terminales, estaciones de trabajo o computadores personales. La conexión de estos usuarios al sitio de la base de datos se realiza a través de hardware de comunicaciones de datos, como las líneas telefónicas, redes de larga distancia, redes locales o dispositivos de comunicaciones por satélite. Muchos sistemas de bases de datos tienen paquetes de comunicaciones que trabajan con el DBMS. El sistema integrado por el DBMS y el sistema de comunicaciones de datos se conoce como sistema **DB/DC**. Además, algunos DBMSs distribuidos se distribuyen físicamente entre varias máquinas. En este caso, son necesarias las redes de comunicaciones para conectar esas máquinas. En ocasiones se trata de **redes de área local** (**LAN**), pero también pueden ser redes de otros tipos.

---

[11] Aunque CASE significa "*computer-aided software engineering*" (ingeniería de software asistida por computador), muchas de las herramientas CASE se utilizan principalmente para el diseño de bases de datos.

2.5 Arquitecturas cliente/servidor centralizadas para los DBMSs

2.5.1 Arquitectura centralizada de los DBMSs

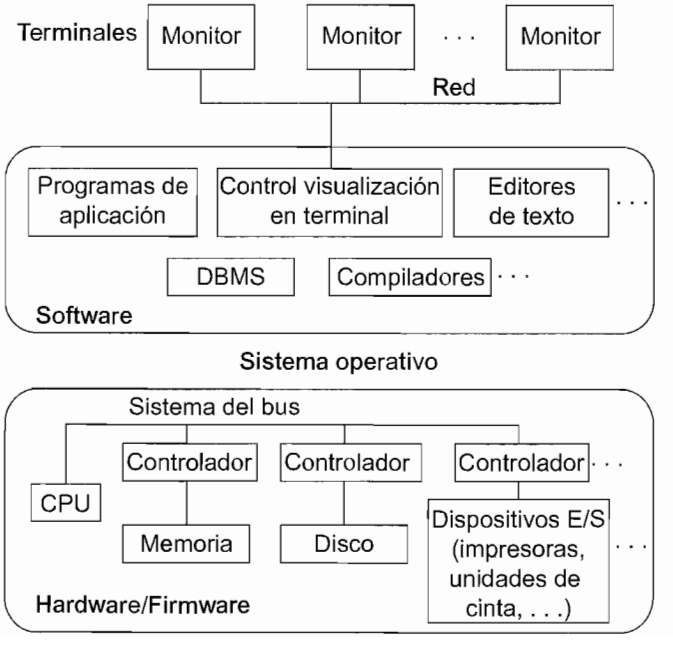
Las arquitecturas de los DBMSs han seguido tendencias parecidas a las arquitecturas de los sistemas de computación generales. Las arquitecturas primigenias utilizaban *mainframes* para proporcionar el procesamiento principal a todas las funciones del sistema, incluyendo las aplicaciones de usuario y los programas de interfaz de usuario, así como a toda la funcionalidad del DBMS. La razón era que la mayoría de los usuarios accedía a esos sistemas a través de terminales de computador que no tenían potencia de procesamiento y sólo ofrecían capacidades de visualización. Por tanto, todo el procesamiento se realizaba remotamente en el sistema computador, y sólo se enviaba la información de visualización y los controles desde el computador a los terminales de visualización, que estaban conectados con el computador central a través de diferentes tipos de redes de comunicaciones.

A medida que bajaban los precios del hardware, la mayoría de los usuarios reemplazaban sus terminales por PCs y estaciones de trabajo. Al principio, los sistemas de bases de datos utilizaban esos computadores de un modo parecido a como utilizaban los terminales de visualización, de modo que el DBMS seguía siendo un DBMS **centralizado** en el que toda la funcionalidad DBMS, ejecución de aplicaciones e interacción con el usuario se llevaba a cabo en una máquina. La Figura 2.4 ilustra los componentes físicos de una arquitectura centralizada. Gradualmente, los sistemas DBMS empezaron a aprovecharse de la potencia de procesamiento disponible en el lado del usuario, lo que llevó a las arquitecturas DBMS cliente/servidor.

2.5.2 Arquitecturas cliente/servidor básicas

En primer lugar vamos a ver la arquitectura cliente/servidor en general, para luego ver cómo se aplica a los DBMSs. La **arquitectura cliente/servidor** se desarrolló para ocuparse de los entornos de computación en los

Figura 2.4. Arquitectura centralizada física.



que una gran cantidad de PCs, estaciones de trabajo, servidores de archivos, impresoras, servidores de bases de datos, servidores web y otros equipos están conectados a través de una red. La idea es definir **servidores especializados** con funcionalidades específicas. Por ejemplo, es posible conectar varios PCs o estaciones de trabajo pequeñas como clientes a un servidor de archivos que mantiene los archivos de las máquinas cliente. Otra máquina puede designarse como **servidor de impresión** conectándola a varias impresoras; después, todas las peticiones de impresión procedentes de los clientes se envían a esta máquina. Los **servidores web** o **servidores de e-mail** también han caído en la categoría de servidores especializados. De este modo, muchas máquinas cliente pueden acceder a los recursos proporcionados por servidores especializados. Las **máquinas cliente** proporcionan al usuario las interfaces apropiadas para utilizar estos servidores, así como potencia de procesamiento local para ejecutar aplicaciones locales. Este concepto se puede llevar al software, donde los programas especializados (como un DBMS o un paquete CAD [diseño asistido por computador]) se almacenan en servidores específicos a los que acceden multitud de clientes. La Figura 2.5 ilustra una arquitectura cliente/servidor en el nivel lógico; la Figura 2.6 es un diagrama simplificado que muestra la arquitectura física. Algunas máquinas sólo serían sitios cliente (por ejemplo, estaciones de trabajo sin discos o estaciones/PCs con discos que sólo tienen instalado el software cliente). Otras máquinas serían servidores dedicados, y otras tendrían funcionalidad de cliente y servidor.

El concepto de arquitectura cliente/servidor asume una estructura subyacente consistente en muchos PCs y estaciones de trabajo, así como una pequeña cantidad de máquinas *mainframe*, conectadas a través de LANs y otros tipos de redes de computadores. En esta estructura, un cliente es normalmente la máquina de un usua-

Figura 2.5. Arquitectura cliente/servidor lógica de dos capas.

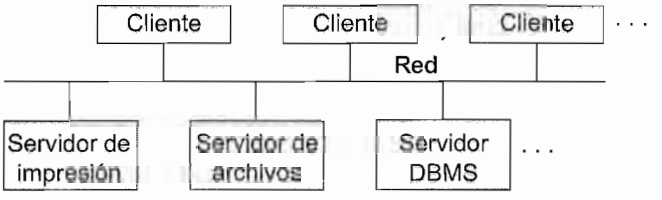
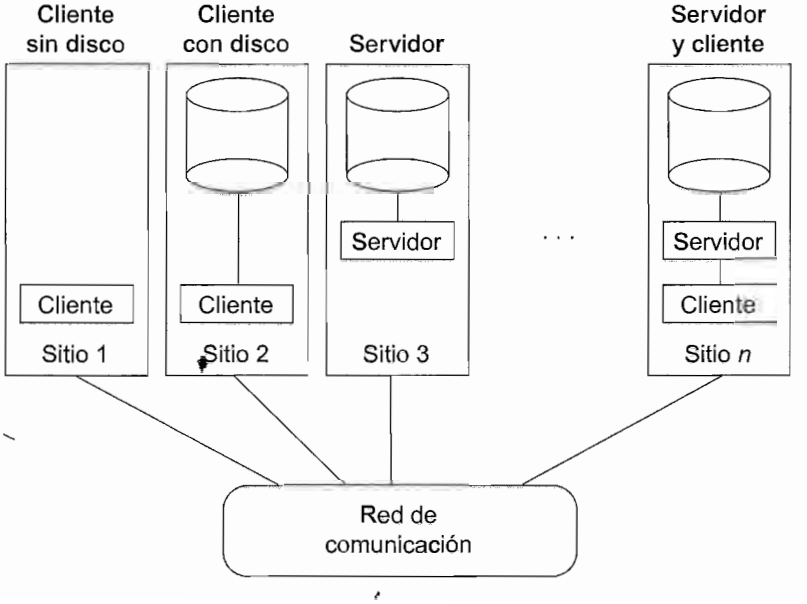


Figura 2.6. Arquitectura cliente/servidor física de dos capas.



rio que proporciona capacidad de interfaz de usuario y procesamiento local. Cuando un cliente requiere acceso a funcionalidad adicional (por ejemplo, acceso a una base de datos) que no existe en esa máquina, conecta con un servidor que ofrece la funcionalidad necesaria. Un servidor es un sistema que contiene hardware y software que pueden proporcionar servicios a los computadores cliente, como acceso a archivos, impresión, archivado o acceso a bases de datos. En el caso general, algunas máquinas sólo instalan el software cliente, mientras otras sólo instalan el software servidor, y otras pueden incluir los dos (véase la Figura 2.6). No obstante, lo más normal es que el software cliente y el software servidor se ejecuten en máquinas separadas. Los dos tipos principales de arquitecturas DBMS básicas se crearon sobre esta estructura cliente/servidor fundamental: dos capas y tres capas.[12] Las explicamos a continuación.

2.5.3 Arquitecturas cliente/servidor de dos capas para los DBMSs

La arquitectura cliente/servidor se está incorporando progresivamente a los paquetes DBMS comerciales. En los sistemas de administración de bases de datos relacionales (RDBMSs), muchos de los cuales empezaron como sistemas centralizados, los primeros componentes del sistema que se movieron al lado del cliente fueron la interfaz de usuario y las aplicaciones. Como SQL (consulte los Capítulos 8 y 9) ofrecía un lenguaje estándar para los RDBMSs, se creó un punto de división lógica entre cliente y servidor. Por tanto, la funcionalidad de consulta y transacción relacionada con el procesamiento SQL permanece en el lado del servidor. En semejante estructura, el servidor se denomina a menudo **servidor de consultas** o **servidor de transacciones** porque proporciona estas dos funcionalidades. En un RDBMS el servidor también se conoce como **servidor SQL**.

En una arquitectura cliente/servidor, los programas de interfaz de usuario y los programas de aplicación se pueden ejecutar en el lado del cliente. Cuando se necesita acceso DBMS, el programa establece una conexión con el DBMS (que se encuentra en el lado del servidor); una vez establecida la conexión, el programa cliente puede comunicarse con el DBMS. El estándar **Conectividad abierta de bases de datos (ODBC, *Open Database Connectivity*)** proporciona una **interfaz de programación de aplicaciones (API, *application programming interface*)**, que permite a los programas del lado del cliente llamar al DBMS, siempre y cuando las máquinas cliente y servidor tengan instalado el software necesario. La mayoría de los fabricantes de DBMSs proporcionan controladores ODBC para sus sistemas. Un programa cliente puede conectar realmente con varios RDBMSs y enviar solicitudes de consulta y transacción utilizando la API ODBC, que después son procesadas en los sitios servidor. Los resultados de una consulta se envían de regreso al programa cliente, que procesará o visualizará los resultados según las necesidades. También se ha definido un estándar relacionado con el lenguaje de programación Java, **JDBC**, que permite a los programas Java cliente acceder al DBMS a través de una interfaz estándar.

Algunos DBMSs orientados a objetos adoptaron la segunda metodología de arquitectura cliente/servidor: los módulos software del DBMS se dividían entre cliente y servidor de un modo más integrado. Por ejemplo, el **nivel servidor** puede incluir la parte del software DBMS responsable de manipular los datos en las páginas del disco, controlar la concurrencia local y la recuperación, almacenar en búfer y caché las páginas de disco, y otras funciones parecidas. Entretanto, el **nivel cliente** puede manipular la interfaz de usuario; las funciones del diccionario de datos; las interacciones DBMS con los compiladores de lenguajes de programación; la optimización de consultas globales, el control de la concurrencia y la recuperación entre varios servidores; la estructuración de objetos complejos a partir de los datos almacenados en los búferes; y otras funciones parecidas. En esta metodología, la interacción cliente/servidor es más estrecha y la realizan internamente los módulos DBMS (algunos de los cuales residen en el cliente y otros en el servidor), en lugar de los usuarios. La división exacta de la funcionalidad varía de un sistema a otro. En semejante arquitectura cliente/servidor, el servidor se llama **servidor de datos** porque proporciona datos de las páginas de disco al cliente. El software DBMS del lado del cliente estructura después estos datos como objetos para los programas cliente.

[12] Hay otras muchas variaciones de las arquitecturas cliente/servidor. Explicaremos las dos más básicas.

Las arquitecturas descritas aquí se llaman **arquitecturas de dos capas** porque los componentes software están distribuidos en dos sistemas: cliente y servidor. Las ventajas de esta arquitectura son su simplicidad y su perfecta compatibilidad con los sistemas existentes. La aparición de la Web cambió los roles de clientes y servidor, lo que condujo a la arquitectura de tres capas.

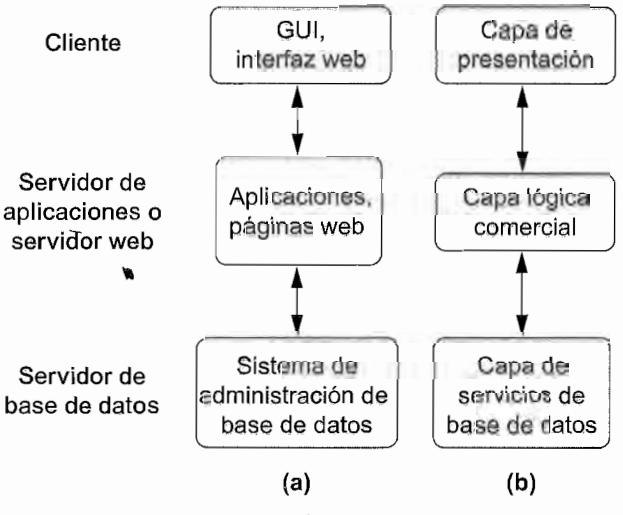
2.5.4 Arquitecturas de tres capas y *n* capas para las aplicaciones web

Muchas aplicaciones web utilizan una arquitectura denominada de tres capas, que añade una capa intermedia entre el cliente y el servidor de la base de datos, como se ilustra en la Figura 2.7(a).

Esta capa **intermedia** se denomina a veces **servidor de aplicaciones** y, en ocasiones, **servidor web**, en función de la aplicación. Este servidor juega un papel intermedio almacenando las reglas comerciales (procedimientos o restricciones) que se utilizan para acceder a los datos del servidor de bases de datos. También puede mejorar la seguridad de la base de datos al comprobar las credenciales del cliente antes de enviar una solicitud al servidor de la base de datos.

Los clientes contienen interfaces GUI y algunas reglas comerciales adicionales específicas de la aplicación. El servidor intermedio acepta solicitudes del cliente, las procesa y envía comandos de bases de datos al servidor de bases de datos, y después actúa como un conducto para pasar datos procesados (parcialmente) desde el servidor de bases de datos a los clientes, donde son procesados de forma más avanzada para su presentación en formato GUI a los usuarios. De este modo, la interfaz de usuario, las reglas de aplicación y el acceso de datos actúan como las tres capas. La Figura 2.7(b) muestra otra arquitectura utilizada por las bases de datos y otros fabricantes de paquetes de aplicaciones. La capa de presentación muestra información al usuario y permite la entrada de datos. La capa lógica comercial manipula las reglas intermedias y las restricciones antes de que los datos sean pasados hacia arriba hasta el usuario, o hacia abajo, hasta el DBMS. La capa inferior incluye todos los servicios de administración de datos. Si la capa inferior está dividida en dos capas (un servidor web y un servidor de bases de datos), entonces tenemos una arquitectura de cuatro capas. Es costumbre dividir las capas entre el usuario y los datos almacenados en componentes aún más sutiles, para de este modo llegar a arquitecturas de *n* capas, donde *n* puede ser cuatro o cinco. Normalmente, la capa lógica comercial está dividida en varias capas. Además de distribuir la programación y los datos por la red, las aplicaciones de *n*

Figura 2.7. Arquitectura cliente/servidor lógica de tres capas, con un par de nomenclaturas comúnmente utilizadas.



capas ofrecen la ventaja de que cualquiera de las capas se puede ejecutar en un procesador adecuado o plataforma de sistema operativo, además de poderse manipular independientemente. Otra capa que los fabricantes de paquetes ERP (planificación de recursos empresariales) y CRM (administración de la relación con el cliente) suelen utilizar es la *capa middleware*, que da cuenta de los módulos *front-end* que comunican con una determinada cantidad de bases de datos *back-end*.

Los avances en la tecnología de cifrado y descifrado hace más segura la transferencia de datos sensibles cifrados desde el servidor hasta el cliente, donde se descifran. Esto último lo puede hacer hardware o software avanzado. Esta tecnología otorga unos niveles altos de seguridad en los datos, aunque los problemas de seguridad en las redes siguen siendo la principal inquietud. Distintas tecnologías de compresión de datos también ayudan a transferir grandes cantidades de datos desde los servidores hasta los clientes a través de redes cableadas e inalámbricas.

2.6 Clasificación de los sistemas de administración de bases de datos

Normalmente se utilizan varios criterios para clasificar los DBMSs. El primero es el **modelo de datos** en el que el DBMS está basado. El **modelo de datos relacional** es el modelo de datos principal que se utiliza en muchos de los DBMSs comerciales actuales. En algunos sistemas comerciales se ha implantado el **modelo de datos de objetos**, pero su uso no se ha extendido. Muchas aplicaciones heredadas todavía se ejecutan en sistemas de bases de datos basados en los **modelos de datos jerárquicos** y **de red**. IMS (IBM) y algunos otros sistemas como System 2K (SAS Ic.) o TDMS son ejemplos de DBMS jerárquicos, que no tuvieron mucho éxito comercial. IMS continúa siendo un actor muy importante entre los DBMSs en uso en instalaciones gubernamentales e industriales, incluyendo hospitales y bancos. Muchos fabricantes utilizaron el modelo de datos de red, y los productos resultantes, como IDMS (Cullinet; ahora, Computer Associates), DMS 1100 (Univac; ahora, Unisys), IMAGE (Hewlett-Packard), VAX-DBMS (Digital; ahora, Compaq) y SUPRA (Cincom) todavía tienen partidarios y sus grupos de usuarios cuentan con organizaciones propias activas. Si a esta lista añadimos el popular sistema de archivos VSAM de IBM, podemos decir que (en el momento de escribir esto) más del 50% de los datos computerizados en todo el mundo se encuentran en estos así llamados **sistemas de bases de datos heredados**.

Los DBMSs relacionales están evolucionando constantemente y, en particular, han ido incorporando muchos de los conceptos que se desarrollaron en las bases de datos de objetos. Esto ha conducido a una nueva clase de DBMSs denominados DBMSs objeto-relacional (objetos relacionales). Los DBMSs se pueden clasificar basándose en el modelo de datos: relacional, objeto, objeto-relacional, jerárquico, de red, y otros.

El segundo criterio que se utiliza para clasificar los DBMSs es el **número de usuarios** soportado por el sistema. Los **sistemas de un solo usuario** sólo soportan un usuario al mismo tiempo y se utilizan principalmente con los PCs. Los **sistemas multiusuario**, que incluyen la mayoría de los DBMSs, soportan varios usuarios simultáneamente.

El tercer criterio es el **número de sitios** sobre los que se ha distribuido la base de datos. Un DBMS es **centralizado** si los datos están almacenados en un solo computador. Un DBMS centralizado puede soportar varios usuarios, pero el DBMS y la base de datos residen en su totalidad en un solo computador. Un **DBMS distribuido** (DDBMS) puede tener la base de datos y el software DBMS distribuidos por muchos sitios, conectados por una red de computadores. Los **DBMSs homogéneos** utilizan el mismo software DBMS en varios sitios. Una tendencia reciente es desarrollar software para acceder a varias bases de datos autónomas preexistentes almacenadas en DBMSs homogéneos. Esto lleva a un **DBMS federado** (o **sistema multibase de datos**), en el que los DBMSs participantes se acoplan y tienen cierto grado de autonomía local. Muchos DBMSs utilizan una arquitectura cliente/servidor.

El cuarto criterio es el coste. Es muy difícil proponer una clasificación de los DBMSs atendiendo al coste. Actualmente, tenemos los productos DBMS de código fuente abierto (gratuito), como MYSQL y PostgreSQL, soportados por terceros con servicios adicionales. Los principales productos RDBMS están disponibles como versiones de prueba válidas para 30 días, así como versiones personales que pueden costar menos de 100 dólares y permitir una funcionalidad considerable. Los sistemas gigantes se están vendiendo de forma modular con componentes para la distribución, duplicación, procesamiento paralelo, capacidad móvil, etcétera; su configuración se realiza mediante la definición de una gran cantidad de parámetros. Además, se venden como licencias (las licencias para un sitio permiten un uso ilimitado del sistema de bases de datos con cualquier número de copias ejecutándose en el sitio cliente). Otro tipo de licencia limita el número de usuarios simultáneos o el número de usuarios sentados en una ubicación. Las versiones para un solo usuario de algunos sistemas como ACCESS se venden por copia, o se incluyen en la configuración global de un escritorio o portátil. Además, el almacenamiento de datos y el minado, así como el soporte de tipos adicionales de datos, tienen un coste adicional. Es muy corriente pagar millones anualmente por la instalación y el mantenimiento de los sistemas de bases de datos.

También podemos clasificar un DBMS según los **tipos de rutas de acceso** para almacenar archivos. Una familia de DBMSs bien conocida está basada en estructuras de archivos inversas. Por último, un DBMS puede ser de **propósito general** o de **propósito especial**. Cuando la principal consideración es el rendimiento, se puede diseñar y construir un DBMS de propósito especial para una aplicación específica; dicho sistema no se puede utilizar para otras aplicaciones sin introducir cambios importantes. Muchos sistemas de reservas en aerolíneas y de directorios telefónicos desarrollados en el pasado son DBMSs de propósito especial. Esto cae en la categoría de los sistemas de **procesamiento de transacciones en línea** (**OLTP**, ***online transaction processing***), que deben soportar una gran cantidad de transacciones simultáneas sin imponer unos retrasos excesivos.

Permítanos elaborar brevemente el criterio principal para clasificar los DBMSs: el modelo de datos. El modelo de datos relacional básico representa una base de datos como una colección de tablas, donde cada tabla se puede almacenar como un archivo separado. La base de datos de la Figura 1.2 parece una representación relacional. La mayoría de las bases de datos relacionales utilizan el lenguaje de consulta de alto nivel SQL y soportan un formato limitado de vistas de usuario. En los Capítulos 5 a 9 explicaremos el modelo relacional, sus lenguajes y operaciones, así como las técnicas para programar aplicaciones relacionales.

El modelo de datos de objetos define una base de datos en términos de objetos, sus propiedades y sus operaciones. Los objetos con la misma estructura y comportamiento pertenecen a una **clase**, y las clases están organizadas en **jerarquías** (o **gráficos acíclicos**). Las operaciones de cada clase son específicas en términos de procedimientos predefinidos denominados **métodos**. Los DBMSs relacionales han ido ampliando sus modelos para incorporar conceptos de bases de datos de objetos y otras capacidades; estos sistemas se conocen como **sistemas objeto-relacional** o **sistemas relacionales extendidos**. En los Capítulos 20 a 22 explicaremos los sistemas de bases de datos de objetos y los sistemas objeto-relacional.

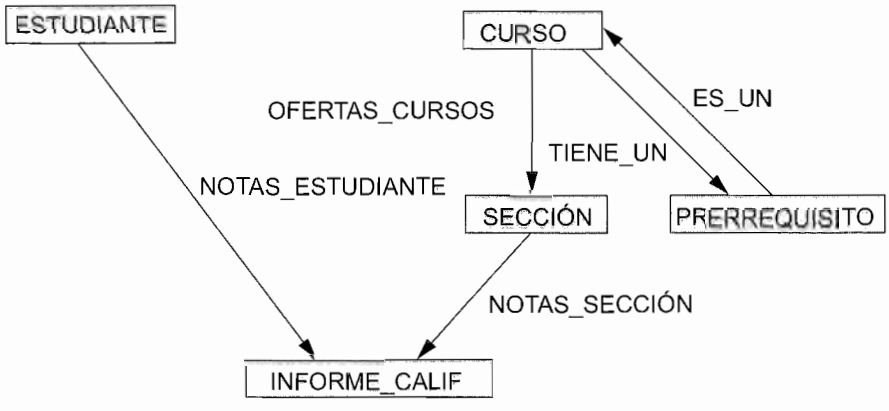
Dos modelos de datos más antiguos e históricamente importantes, ahora conocidos como modelos de datos heredados, son los modelos de red y jerárquico. El **modelo de red** representa los datos como tipos de registros, y también representa un tipo limitado de relación 1:N, denominado **tipo conjunto**. Una relación 1:N, o uno-a-muchos, relaciona una instancia de un registro con muchas instancias de registro mediante algún mecanismo de punteros en esos modelos. La Figura 2.8 muestra el diagrama del esquema de una red para la base de datos de la Figura 1.2, donde los tipos de registros se muestran como rectángulos y los tipos conjunto como flechas de dirección etiquetadas. El modelo de red, también conocido como modelo CODASYL DBTG,[13] tiene un lenguaje *record-at-a-time* asociado que debe incrustarse en un lenguaje de programación *host*. El DML de red se propuso en el informe Database Task Group (DBTG) de 1971 como una extensión del

[13] CODASYL DBTG significa "*Conference on Data Systems Languages Database Task Group*", que es el comité que especificó el modelo de red y su lenguaje.

lenguaje COBOL. Proporciona comandos para localizar registros directamente (por ejemplo, FIND ANY <*tipo-registro*> USING <*lista-campos*>, o FIND DUPLICATE <*tipo-registro*> USING <*lista-campos*>). Tiene comandos para soportar conversiones dentro de los tipos conjunto (por ejemplo, GET OWNER, GET {FIRST, NEXT, LAST} MEMBER WITHIN <*tipo-conjunto*> WHERE <*condición*>). También tiene comandos para almacenar datos nuevos (por ejemplo, STORE <*tipo-registro*>) y convertirlos en parte de un tipo conjunto (por ejemplo, CONNECT <*tipo-registro*> TO <*tipo-conjunto*>). El lenguaje también manipula muchas consideraciones adicionales como los tipos de registro y los tipos conjunto, que están definidos por la posición actual del proceso de navegación dentro de la base de datos. Actualmente, lo utilizan principalmente los DBMSs IDMS, IMAGE y SUPRA. El **modelo jerárquico** representa los datos como estructuras en forma de árboles jerárquicos. Cada jerarquía representa una cantidad de registros relacionados. No hay ningún lenguaje estándar para el modelo jerárquico. Un DML jerárquico popular es DL/1 del sistema IMS, que dominó el mercado de los DBMSs durante más de 20 años, entre 1965 y 1985, y es un DBMS que incluso hoy en día sigue utilizándose ampliamente, manteniendo un porcentaje muy alto de datos en bases de datos gubernamentales, sanitarias, bancarias y aseguradoras. Su DML, denominado DL/1, fue el estándar industrial de facto durante mucho tiempo. DL/1 tiene comandos para localizar un registro (por ejemplo, GET {UNIQUE, NEXT} <*tipo-registro*> WHERE <*condición*>). También ofrece funciones de navegación para navegar por las jerarquías (por ejemplo, GET NEXT WITHIN PARENT o GET {FIRST, NEXT} PATH <*especificación-ruta-jerárquica*> WHERE <*condición*>). Tiene los medios apropiados para almacenar y actualizar registros (por ejemplo, INSERT <*tipo-registro*>, REPLACE <*tipo-registro*>). En los Apéndices E y F veremos una breve panorámica de los modelos de red y jerárquico.[14]

El **modelo de lenguaje de marcado extendido (XML, *eXtended Markup Language*)**, ahora considerado el estándar para el intercambio de datos por Internet, también utiliza estructuras en forma de árbol jerárquico. Combina los conceptos de bases de datos con conceptos procedentes de los modelos de representación de documentos. Los datos se representan como elementos; con el uso de etiquetas, los datos se pueden anidar para crear estructuras jerárquicas complejas. Este modelo se parece conceptualmente al modelo de objetos, pero utiliza una terminología diferente. En el Capítulo 27 explicaremos XML y veremos cómo se relaciona con las bases de datos.

Figura 2.8. El esquema de la Figura 2.1 en la notación del modelo de red.



[14] Los capítulos completos sobre los modelos de red y jerárquico de la segunda edición de este libro están disponibles en el sitio web complementario: www.librosite.net/elmasri.

2.7 Resumen

En este capítulo hemos introducido los principales conceptos que se utilizan en los sistemas de bases de datos. Hemos definido un modelo de datos y distinguido tres categorías principales:

- Modelos de datos de alto nivel o conceptuales (basados en entidades y relacionales).
- Modelos de datos de nivel bajo o físicos.
- Modelos de datos representativos o de implementación (basados en registros, orientados a objetos).

Distinguimos el esquema, o descripción de una base de datos, de la propia base de datos. El esquema no cambia muy a menudo, mientras que el estado de la base de datos cambia cada vez que se insertan, eliminan o modifican datos. Después describimos la arquitectura DBMS de tres esquemas, que permite tres niveles de esquema:

- Un esquema interno describe la estructura del almacenamiento físico de la base de datos.
- Un esquema conceptual es una descripción de nivel alto de toda la base de datos.
- Los esquemas externos describen las vistas de los diferentes grupos de usuarios.

Un DBMS que separa limpiamente los tres niveles debe tener mapeados entre los esquemas para transformar las solicitudes y los resultados de un nivel al siguiente. La mayoría de los DBMSs no separan completamente los tres niveles. Utilizamos la arquitectura de tres esquemas para definir los conceptos de la independencia lógica y física de datos.

Después explicamos los principales tipos de lenguajes e interfaces que los DBMSs soportan. Se utiliza un lenguaje de definición de datos (DDL) para definir el esquema conceptual de la base de datos. En la mayoría de los DBMSs, el DDL también define las vistas de los usuarios y, algunas veces, las estructuras de almacenamiento; en otros DBMSs, pueden existir lenguajes separados (VDL, SDL) para especificar las vistas y las estructuras de almacenamiento. Esta distinción desaparece en las implementaciones relacionales actuales con SQL haciendo el papel de lenguaje común para muchos papeles, como la definición de vistas. La parte de definición del almacenamiento (SDL) se incluyó en muchas versiones de SQL, pero ahora ha quedado relegado a comandos especiales para el DBA en los DBMSs relacionales. El DBMS compila todas las definiciones de esquema y almacena sus descripciones en el catálogo del DBMS. Se utiliza un lenguaje de manipulación de datos (DML) para especificar las recuperaciones y las actualizaciones de la base de datos. Los DMLs pueden ser de alto nivel (orientados a conjunto, no procedimentales) o de bajo nivel (orientados a registros, procedimentales). Un DML de alto nivel se puede incrustar en un lenguaje *host*, y también se puede utilizar como un lenguaje independiente; en el último caso se denomina con frecuencia lenguaje de consulta.

Hemos explicado los diferentes tipos de interfaces proporcionadas por los DBMSs, así como los tipos de usuarios de DBMS con los que cada interfaz está asociada. Después, hemos explicado el entorno del sistema de bases de datos, los módulos software DBMS típicos y las utilidades DBMS destinadas a ayudar a los usuarios y al DBA en sus tareas. Continuamos con una panorámica de las arquitecturas de dos y tres capas para las aplicaciones de bases de datos, moviéndonos progresivamente hasta la arquitectura de *n* capas, mucho más común en la mayoría de las aplicaciones modernas, particularmente las basadas en la Web.

Por último, clasificamos los DBMS según varios criterios: modelo de datos, número de usuarios, número de sitios, tipos de rutas de acceso y en general. Hablamos de la disponibilidad de los DBMSs y de los módulos adicionales (desde la ausencia de coste en forma de software de código abierto, hasta las configuraciones con un coste anual de varios millones por mantenimiento). También nos referimos a la variedad de acuerdos de licencia para los DBMSs y los productos relacionados. La principal clasificación de los DBMSs está basada en los modelos de datos. Explicamos brevemente los principales modelos que se utilizan en los DBMSs comerciales actuales y ofrecimos un ejemplo de modelo de datos de red.

Preguntas de repaso

2.1. Defina los siguientes términos: modelo de datos, esquema de base de datos, estado de la base de datos, esquema interno, esquema conceptual, esquema externo, independencia de datos, DDL, DML, SDL, VDL, lenguaje de consulta, lenguaje host, sublenguaje de datos, utilidad de base de datos, catálogo, arquitectura cliente/servidor, arquitectura de tres capas y arquitectura de *n* capas.

2.2. Explique las principales categorías de modelos de datos.

2.3. ¿Cuál es la diferencia entre un esquema de base de datos y un estado de base de datos?

2.4. Describa la arquitectura de tres esquemas. ¿Por qué son necesarios los mapeos entre los niveles de esquema? ¿Cómo soportan los diferentes lenguajes de definición de esquemas esta arquitectura?

2.5. ¿Cuál es la diferencia entre la independencia lógica de datos y la independencia física de datos? ¿Cuál es más difícil de conseguir? ¿Por qué?

2.6. ¿Cuál es la diferencia entre DMLs procedimentales y no procedimentales?

2.7. Explique los diferentes tipos de interfaces amigables con el usuario y los tipos de usuarios que normalmente utilizan cada una de ellas.

2.8. ¿Con qué otro software de computación interactúa un DBMS?

2.9. ¿Cuál es la diferencia entre las arquitecturas cliente/servidor de dos y tres capas?

2.10. Explique algunos tipos de utilidades y herramientas de bases de datos, así como sus funciones.

2.11. ¿Cuál es la funcionalidad adicional que se incorpora en la arquitectura de *n* capas ($n > 3$)?

Ejercicios

2.12. Piense en diferentes usuarios para la base de datos de la Figura 1.2. ¿Qué tipos de aplicaciones necesitaría cada uno? ¿A qué categoría de usuario pertenecería cada uno y qué tipo de interfaz necesitaría cada uno de ellos?

2.13. Elija una aplicación de bases de datos con la que esté familiarizado. Diseñe un esquema y muestre una base de datos de ejemplo para esa aplicación, utilizando la notación de las Figuras 1.2 y 2.1. ¿Qué tipos de información adicional y de restricciones tendrá que representar en el esquema? Piense en los distintos usuarios de su base de datos y diseñe una vista para cada uno.

2.14. Si estuviera diseñando un sistema basado en la Web para realizar reservas en aerolíneas y vender billetes de avión, ¿qué arquitectura DBMS elegiría de las presentadas en la Sección 2.5? ¿Por qué? ¿Por qué las demás arquitecturas no serían una buena elección?

2.15. Considere la Figura 2.1. Además de las restricciones que relacionan los valores de las columnas de una tabla con las columnas de otra, también hay otras restricciones que restringen los valores de una columna o de una combinación de columnas de una tabla. Una de esas restricciones estipula que una columna o un grupo de columnas debe ser único a través de todas las filas de la tabla. Por ejemplo, en la tabla ESTUDIANTE, la columna NumEstudiante debe ser única (para evitar que dos estudiantes diferentes tengan el mismo NumEstudiante). Identifique la columna o el grupo de columnas de las otras tablas que deben ser únicos por todas las filas de la tabla.

Bibliografía seleccionada

Muchos libros de bases de datos, incluyendo Date (2004), Silberschatz y otros (2005), Ramakrishnan and Gehrke (2003), García-Molina y otros (2000, 2002), y Abiteboul y otros (1995), ofrecen una explicación de los diferentes conceptos de bases de datos aquí explicados. Tsichritzis and Lochovsky (1982) es un libro antiguo dedicado a los modelos de datos. Tsichritzis and Klug (1978) y Jardine (1977) presentan la arquitectura de tres esquemas, que se sugirió por primera vez en el informe DBTG CODASYL (1971) y, posteriormente, en un informe del American National Standards Institute (ANSI) en 1975. En Codd (1992) se ofrece un aná-

lisis en profundidad del modelo de datos relacional y de algunas de sus posibles extensiones. El estándar propuesto para las bases de datos orientadas a objetos se describe en Cattell y otros (2000). En la Web hay muchos documentos que describen XML, como, por ejemplo, XML (2005).

ETI Extract Toolkit (*http://www.eti.com*) y la herramienta de administración de bases de datos, DB Artisan, de Embarcadero Technologies (*http://www.embarcadero.com*), son ejemplos de utilidades de bases de datos.