INTRODUCCIÓN A LOS Sistemas de bases de datos

SÉPTIMA EDICIÓN

C. J. Date

TRADUCCIÓN:
I. Q. Sergio Luis María Ruiz Faudón
Ingeniero Químico, Analista de Sistemas
Sergio Kourchenko Barrena

REVISIÓN TÉCNICA:
Dr. Felipe López Gamino
Instituto Tecnológico Autónomo de México

Pearson Educación ®

MÉXICO • ARGENTINA • BRASIL • COLOMBIA • COSTA RICA • CHILE
ESPAÑA • GUATEMALA • PERÚ • PUERTO RICO • VENEZUELA

Contenido

CAPITULO 1

Panorama general de la administración de bases de datos

1.1 INTRODUCCIÓN

Un sistema de **bases de datos** es básicamente un *sistema computarizado para llevar registros.* Es posible considerar a la propia **base de datos** como una especie de armario electrónico para archivar; es decir, es un depósito o contenedor de una colección de archivos de datos computarizados. Los usuarios del sistema pueden realizar una variedad de operaciones sobre dichos archivos. por ejemplo:

- Agregar nuevos archivos vacíos a la base de datos;
- Insertar datos dentro de los archivos existentes;
- Recuperar datos de los archivos existentes;
- Modificar datos en archivos existentes;
- Eliminar datos de los archivos existentes;
- Eliminar archivos existentes de la base de datos.

La figura 1.1 muestra una base de datos reducida que contiene un solo archivo, denominado CAVA, el cual contiene a su vez datos concernientes al contenido de una cava de vinos. La

NICHO#	VINO	PRODUCTOR	AÑO	BOTELLAS	LISTO
2	Chardonnay	Buena Vista	1997	1	1999
3	Chardonnay	Geyser Peak	1997	5	1999
6	Chardonnay	Simi	1996	4	1996
12	Joh. Riesling	Jekel	1998	1	1999
21	Fumé Blanc	Ch. St. Jean	1997	4	1999
22	Fumé Blanc	Robt. Mondavi	1996	2	1998
30	Gewurztraminer	Ch. St. Jean	1998	3	1999
43	Cab. Sauvignon	Windsor	1991	12	2000
45	Cab. Sauvignon	Geyser Peak	1994	12	2002
48	Cab. Sauvignon	Robt. Mondavi	1993	12	2004
50	Pinot Noir	Gary Farrel	1996	3	1999
51	Pinot Noir	Fetzer	1993	3	2000
52	Pinot Noir	Dehlinger	1995	2	1998
58	Merlot	Clos du Bois	1994	9	2000
64	Zinfandel	Cline	1994	9	2003
72	Zinfandel	Rafanelli	1995	2	2003

Figura 1.1 La base de datos de la cava de vinos (archivo CAVA).

Recuperación:

```
SELECT VINO, NICHO#, PRODUCTOR
FROM CAVA
WHERE LISTO = 2000
```

Resultado (por ejemplo, como se muestra en una pantalla de monitor):

VINO	NICHO#	PRODUCTOR
Cab. Sauvignon	43	Windsor Fetzer
Pinot Noir	51	Clos du Bois
Merlot	58	

Figura 1.2 Ejemplo de recuperación.

figura 1.2 muestra una operación de **recuperación** desde la base de datos, junto con los datos devueltos por dicha operación. *Nota:* Para una mayor claridad, a lo largo del libro mostramos en mayúsculas las operaciones de base de datos, los nombres de archivo y otro material similar. En la práctica es a menudo más conveniente escribir este material en minúsculas. La mayoría de los sistemas aceptan ambas denominaciones.

La figura 1.3 muestra ejemplos de operaciones de **inserción, modiñcación y eliminación** de la base de datos anterior que prácticamente se explican por sí mismos. Más adelante, en los capítulos 3,4,5, y en algunas otras partes, proporcionaré ejemplos de la incorporación y eliminación de archivos completos.

Inserción de datos nuevos:

```
INSERT
INTO CAVA ( NICHO*, VINO, PRODUCTOR, AÑO, BOTELLAS, LISTO )
VALUES ( 53, 'Pinot Noir', 'Saintsbury', 1997, 6, 2001 ) ;
```

Modificación de datos existentes:

```
UPDATE CAVA
SET   BOTELLAS • 4
WHERE NICHO# = 3 ;
```

Eliminación de datos existentes:

```
DELETE
FROM CAVA
WHERE NICHO# = 2 ;
```

Figura 1.3 Ejemplos de inserción, modificación y eliminación.

Puntos importantes de estos ejemplos:

1. Por razones obvias, a los archivos computarizados como el de CAVA de la figura 1.1.a menudo se les llama **tablas** (con más precisión, tablas **relaciónales.** Vea las secciones 1.3 y 1-6).
2. Podemos pensar en las **filas** de dicha tabla como los registros del archivo y en las **colum nas** como los campos de dichos registros. En este libro, emplearemos la terminología de registros y campos cuando hablemos de sistemas de base de datos en general (principal mente en los dos primeros capítulos); usaremos la terminología de filas y columnas cuando hablemos de sistemas relaciónales específicos (nuevamente, vea las secciones 1.3 y 1.6). *Nota:* En realidad, cuando abordemos explicaciones más formales en las partes posteriores del libro, cambiaremos a términos más formales.
3. Por razones de simplicidad, en el ejemplo anterior hicimos la suposición tácita de que las columnas VINO y PRODUCTOR contienen datos de tipo cadena de caracteres y que las demás columnas contienen datos enteros. Consideraremos con más detalle la cuestión de los **tipos de datos** de las columnas en los capítulos 3, 4 y en particular en el 5.
4. La columna NICHO# constituye la **clave primaria** de la tabla CAVA (lo que significa que no es posible que dos filas de CAVA contengan el mismo valor de NICHO#). A me nudo usamos un subrayado doble para señalar las columnas de clave primaria, como en la figura 1.1.
5. Las operaciones de ejemplo o "instrucciones" de las figuras 1.2 y 1.3 —SELECT, **INSERT, UPDATE, DELETE**— están expresadas en un lenguaje denominado.SQL. SQL es el len guaje estándar para interactuar con bases de datos relaciónales y es soportado por práctica mente todos los productos de base de datos actuales. *Nota:* El nombre "SQL" significaba originalmente "Lenguaje estructurado de consultas" y se pronunciaba "sikuel". Sin embar go, ahora que el lenguaje se ha convertido en un estándar, el nombre es solamente repre sentativo —no es oficialmente la abreviatura de nada— y la balanza se inclinó en favor de la pronunciación "es-kiu-el". En el libro tomaremos esta última pronunciación.
6. Observe que SQL utiliza la palabra clave UPDATE para indicar específicamente un "cam bio". Este hecho puede causar confusión, debido a que este término también solía referirse a las tres operaciones INSERT, UPDATE y DELETE como grupo. En este libro, usaremos la palabra "actualizar", en minúsculas, cuando nos refiramos al significado genérico y el operador UPDATE, en mayúsculas, cuando se trate de la operación específica de modi ficación.
7. Como es probable que ya sepa, la gran mayoría de sistemas de bases de datos actuales son relaciónales (o de todos modos deberían serlo —vea el capítulo 4, sección 4.7). En parte por esta razón, este libro hace énfasis en dichos sistemas.

Una última observación preliminar: la comprensión del material de este capítulo y el siguiente es fundamental para una apreciación completa de las características y capacidades de un sistema moderno de base de datos. Sin embargo, no puede negarse que el material es en cierto modo abstracto y un poco árido en ciertas partes, y que tiende a abarcar un gran número de conceptos y términos que podrían ser nuevos para usted. En las partes posteriores del libro —en especial en los capítulos 3 y 4— encontrará material mucho menos abstracto y por lo tanto quizás más comprensible. De ahí que tal vez prefiera, por el momento, dar a estos dos primeros

capítulos una "leída ligera" y volverlos a leer con más detenimiento cuando sean más relevantes para los temas que esté abordando.

1.2 ¿QUÉ ES UN SISTEMA DE BASE DE DATOS?

Para repetir lo que mencionamos en la sección anterior, un sistema de base de datos es básicamente un sistema computarizado para guardar registros; es decir, es un sistema computarizado cuya finalidad general es almacenar información y permitir a los usuarios recuperar y actualizar esa información con base en peticiones. La información en cuestión puede ser cualquier cosa que sea de importancia para el individuo u organización; en otras palabras, todo lo que sea necesario para auxiliarle en el proceso general de su administración.

Nota: en este libro los términos "datos" e "información" los trato como sinónimos. Algunos autores prefieren distinguir entre ambos, utilizando "datos" para referirse a lo que está en realidad almacenado en la base de datos e "información" para referirse al *significado* de esos datos como lo entiende algún usuario. La diferencia es importante; tan importante que parece preferible hacerla explícita donde sea necesario, en vez de depender de una diferenciación un tanto arbitraria entre dos términos que son en esencia sinónimos.

La figura 1.4 es una imagen simplificada de un sistema de base de datos. Pretende mostrar que un sistema de base de datos comprende cuatro componentes principales: **datos, hardware, software y usuarios.** A continuación consideramos brevemente estos cuatro componentes. Por

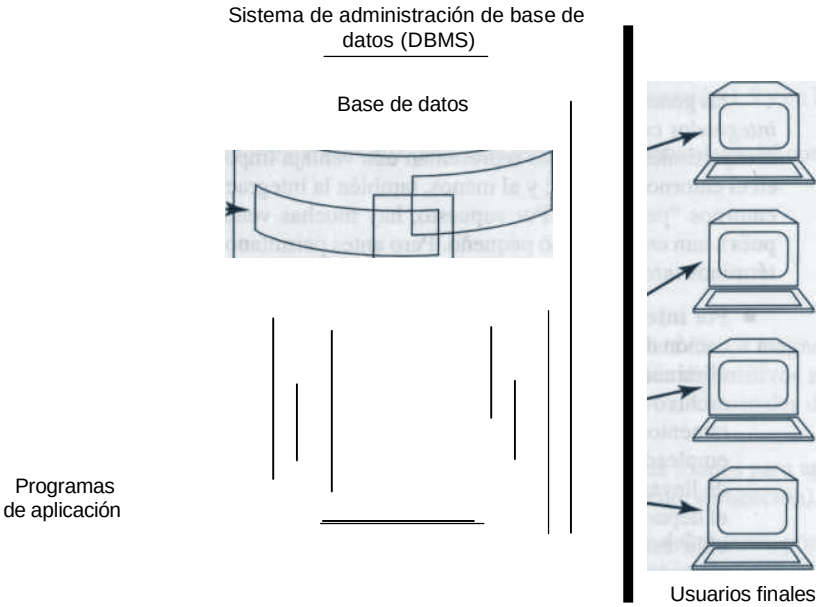


Figura 1.4 Imagen simplificada de un sistema de base de datos.

supuesto, más delante explicaremos cada uno con más detalle (con excepción del componente de hardware, cuyos detalles exceden en su mayoría el alcance de este libro).

Datos

Los sistemas de bases de datos están disponibles en máquinas que van desde las computadoras personales más pequeñas hasta las mainframes más grandes. Sobra decir que las facilidades que proporciona un sistema están determinadas hasta cierto punto por el tamaño y potencia de la máquina subyacente. En particular, los sistemas que se encuentran en máquinas grandes (s i s temas grandes") tienden a ser *multiusuario,* mientras que los que se ejecutan en máquinas pe queñas ("sistemas pequeños") tienden a ser *de un solo usuario.* Un **sistema de un solo usuario** es aquel en el que sólo un usuario puede tener acceso a la base de datos en un momento dado; un **sistema multiusuario** es aquel en el cual múltiples usuarios pueden tener acceso simultáneo a la base de datos. Como sugiere la figura 1.4, en este libro generalmente tomaremos el último caso; aunque de hecho la distinción es irrelevante en lo que respecta a la mayoría de los usuarios: En general, el objetivo principal en los sistemas multiusuario es precisamente permitir que cada usuario se comporte como si estuviera trabajando en un sistema *de un solo usuario.* **Los** problemas especiales de los sistemas multiusuario son en su mayoría problemas internos del sistema y no aquellos que son visibles al usuario (vea la parte IV de este libro, en especial el capítulo 15).

Nota: Para efectos de simplicidad, es conveniente suponer que la totalidad de los datos del sistema está almacenada en una sola base de datos, y en este libro haremos constantemente esta suposición (ya que materialmente no afecta ninguna de nuestras explicaciones posteriores). Sir embargo, en la práctica podría haber buenas razones (incluso en un sistema pequeño) para que los datos sean separados en diferentes bases de datos. Abordaremos algunas de estas razones mas adelante, en el capítulo 2 y en otras secciones.

En general, los datos de la base de datos —por lo menos en un sistema grande— serán tanto *integrados* como *compartidos.* Como veremos en la sección 1.4, los aspectos de integración y compartimiento de datos representan una ventaja importante de los sistemas de bases de datos en el entorno "grande"; y al menos, también la integración de datos puede ser importante en los entornos "pequeños". Por supuesto, hay muchas ventajas adicionales (que abordaremos después), aun en el entorno pequeño. Pero antes permítanos explicar lo que queremos decir con 1 términos *integrado* y *compartido.*

- Por **integrada,** queremos decir que podemos imaginar a la base de datos como una unifi-
  cación de varios archivos que de otro modo serían distintos, con una redundancia entre ellos eliminada al menos parcialmente. Por ejemplo, una base de datos dada podría contener un archivo EMPLEADO que proporcionara los nombres de los empleados, domicilios, depar tamentos, sueldos, etc. y un archivo INSCRIPCIÓN que representara la inscripción de los empleados a los cursos de capacitación (consulte la figura 1.5). Suponga ahora que, a fu de llevar a cabo el proceso de administración de cursos de capacitación, es necesario saber el departamento de cada estudiante inscrito. Entonces, resulta claro que no es necesario in cluir esa información de manera redundante en el archivo INSCRIPCIÓN, debido a que siempre puede consultarse haciendo referencia al archivo EMPLEADO.
- Por **compartida,** queremos decir que las piezas individuales de datos en la base pueden s compartidas entre diferentes usuarios y que cada uno de ellos puede tener acceso a la misma pieza de datos, probablemente con fines diferentes. Como indiqué anteriormente, distintos

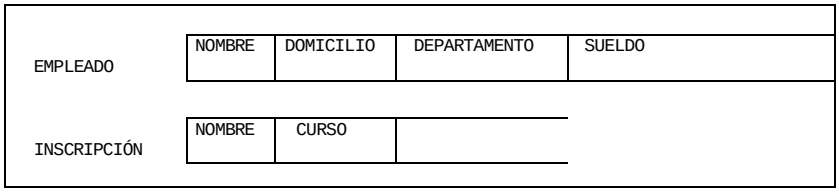


Figura 1.5 Los archivos EMPLEADO e INSCRIPCIÓN.

usuarios pueden en efecto acceder a la misma pieza de datos *al mismo tiempo* ("acceso concurrente"). Este compartimiento, concurrente o no, es en parte consecuencia del hecho de que la base de datos está integrada. En el ejemplo citado arriba, la información de departamento en el archivo EMPLEADO sería típicamente compartida por los usuarios del Departamento de personal y los usuarios del Departamento de capacitación; y como ya sugerí, estas dos clases de usuarios podrían emplear esa información para fines diferentes. *Nota:* en ocasiones, si la base de datos *no es* compartida, se le conoce como "personal" o como "específica de la aplicación".

Otra consecuencia de los hechos precedentes —que la base de datos sea integrada y (por lo regular) compartida— es que cualquier usuario ocupará normalmente sólo una pequeña parte de la base de datos total; lo que es más, las partes de los distintos usuarios se traslaparán de diversas formas. En otras palabras, una determinada base de datos será percibida de muchas formas diferentes por los distintos usuarios. De hecho, aun cuando dos usuarios tengan la misma porción de la base de datos, su visión de dicha parte podría diferir considerablemente a un nivel detallado. Este último punto lo explico en forma más completa en la sección 1.5 y en los capítulos 2, 3 y en especial el 9.

En la sección 1.3 tendremos más qué decir con respecto a la naturaleza del componente de datos del sistema.

Hardware

Los componentes de hardware del sistema constan de:

- Los volúmenes de almacenamiento secundario —principalmente discos magnéticos— que se emplean para contener los datos almacenados, junto con los dispositivos asociados de E/S (unidades de discos, etc.), los controladores de dispositivos, los canales de E/S, entre otros; y
- Los procesadores de hardware y la memoria principal asociada usados para apoyar la eje cución del software del sistema de base de datos (vea la siguiente subsección).

Este libro no hace mucha referencia a los aspectos de hardware del sistema, por las siguientes razones (entre otras): Primero, estos aspectos conforman un tema importante por sí mismos; segundo, los problemas que se encuentran en esta área no son exclusivos de los sistemas de base de datos; y tercero, dichos problemas han sido investigados en forma minuciosa y descritos en otras partes.

Software

Entre la base de datos física —es decir, los datos como están almacenados físicamente— usuarios del sistema, hay una capa de software conocida de manera indistinta como el administrador de base de datos o el servidor de base de datos; o más comúnmente como el sistema de administración de base de datos (DBMS). Todas las solicitudes de acceso a la base de datos son manejadas por el DBMS; las características que esbozamos en la sección 1.1 para agre eliminar archivos (o tablas), recuperar y almacenar datos desde y en dichos archivos, etcétera son características que proporciona el DBMS. Por lo tanto, una función general que ofrece DBMS consiste en *ocultar a los usuarios de la base de datos los detalles al nivel de hará* (en forma muy parecida a como los sistemas de lenguajes de programación ocultan a los programadores de aplicaciones los detalles a nivel de hardware). En otras palabras, el DBMS ofrece a los usuarios una percepción de la base de datos que está, en cierto modo, por encima del nivel del hardware y que maneja las operaciones del usuario (como las operaciones SQL explicad brevemente en la sección 1.1) expresadas en términos de ese nivel más alto de percepción. A lo largo de este libro explicaremos con mayor detalle ésta y otras funciones del DBMS. Algunos aspectos adicionales:

- El DBMS es, por mucho, el componente de software más importante del sistema en gene ral, aunque no es el único. Otros comprenden las utilerías, herramientas de desarrollo de aplicaciones, ayudas de diseño, generadores de informes y (el más importante) el *administrador de transacciones* o *monitor PT.* Para una mayor explicación de estos componentes. consulte los capítulos 2 y 3 y (en especial) la parte IV.
- El término *DBMS* se usa también para referirse en forma genérica a un producto determin de algún fabricante; por ejemplo, el producto "DB2 Universal Database" de IBM para OS/390. El término *ejemplar de DBMS* se usa entonces para referirse a una copia de dicho producto que opera en alguna instalación de computadora determinada. Como seguram notará, en ocasiones es necesario distinguir cuidadosamente entre estos dos conceptos.

  Nota: Debemos advertirle que en la industria de las computadoras la gente a menudo usa el término *base de datos* cuando en realidad se refieren al *DBMS* (en cualquiera de lo sentidos anteriores). He aquí un ejemplo típico: "El fabricante de la base de datos X superó al fabricante de la base de datos *Y* en proporción de dos a uno." Este uso es engañoso y no es correcto, aunque es mucho muy común. (Por supuesto, el problema es que si al DBMS lo llamamos base de datos, entonces ¿cómo llamaremos a la base de datos?) *Advertencia para el lector.*

Usuarios

Consideramos tres grandes clases de usuarios (y que en cierto modo se traslapan):

- Primero, hay programadores de aplicaciones responsables de escribir los programas de aplicación de base de datos en algún lenguaje de programación como COBOL. PL/1. C++ Java o algún lenguaje de alto nivel de la "cuarta generación" (vea el capítulo 2). Estos pro-gramas acceden a la base de datos emitiendo la solicitud apropiada al DBMS (por lo regu lar una instrucción SQL). Los programas en sí pueden ser aplicaciones convenciona por lotes o pueden ser aplicaciones en línea, cuyo propósito es permitir al usuario final el acceso a la base de datos desde una estación de trabajo o terminal en línea (vea el parrafo siguiente). Las aplicaciones más modernas pertenecen a esta variedad.

- En consecuencia, la segunda clase de usuarios son los **usuarios finales,** quienes interactúan con el sistema desde estaciones de trabajo o terminales en línea. Un usuario final puede acceder a la base de datos a través de las aplicaciones en línea mencionadas en el párrafo anterior, o bien puede usar una interfaz proporcionada como parte integral del software del sistema de base de datos. Por supuesto, las interfaces proporcionadas por el fabricante están apoyadas también por aplicaciones en línea, aunque esas aplicaciones están **integradas;** es decir, no son escritas por el usuario. La mayoría de los sistemas de base de datos incluyen por lo menos una de estas aplicaciones integradas, digamos un **procesador de lenguaje de consulta,** mediante el cual el usuario puede emitir solicitudes a la base de datos (también conocidas como instrucciones o *comandos),* como SELECT e INSERT, en forma interactiva con el DBMS. El lenguaje SQL mencionado en la sección 1.1 es un ejemplo típico de un lenguaje de consulta de base de datos.

  Nota: El término "lenguaje de consulta", a pesar de ser común, no es muy preciso, ya que el verbo "consultar" en lenguaje normal sugiere sólo una *recuperación,* mientras que los lenguajes de consulta por lo regular (aunque no siempre) ofrecen también actualización y otras operaciones.

  La mayoría de los sistemas proporcionan además interfaces integradas adicionales en las que los usuarios no emiten en absoluto solicitudes explícitas a la base de datos, como SELECT, sino que en vez de ello operan mediante (por ejemplo) la selección de elementos en un menú o llenando casillas de un formulario. Estas interfaces **controladas por menus** o **por formularios** tienden a facilitar el uso a personas que no cuentan con una capacitación formal en IT (Tecnología de la información; la abreviatura IS, de Sistemas de información, también es muy usada con el mismo significado). En contraste, las **interfaces controladas por comandos** (por ejemplo, los lenguajes de consulta) tienden a requerir cierta experiencia profesional en IT, aunque tal vez no demasiada (obviamente no tanta como la que es necesaria para escribir un programa de aplicación en un lenguaje como COBOL). Por otra parte, es probable que una interfaz controlada por comandos sea más flexible que una controlada por menus o por formularios, dado que los lenguajes de consulta por lo regular incluyen ciertas características que no manejan esas otras interfaces.
- El tercer tipo de usuario, que no aparece en la figura 1.4, es el **administrador de base de datos** o DBA. La función del DBA, y la función asociada (muy importante) del administrador de **datos,** se abordará en la sección 1.4 y en el capítulo 2 (sección 2.7).

Con esto concluimos nuestra descripción preliminar de los aspectos más importantes de un sistema de base de datos. Ahora continuaremos con la explicación de estas ideas con un poco más de detalle.

1.3 ¿QUE ES UNA BASE DE DATOS?

Datos persistentes

Es una costumbre referirse a los datos de la base de datos como "persistentes" (¡aunque en realidad éstos podrían no persistir por mucho tiempo!). Por *persistentes* queremos decir, de manera intuitiva, que el tipo de datos de la base de datos difiere de otros datos más efímeros, como los datos de entrada, los datos de salida, las instrucciones de control, las colas de trabajo, los bloques de control de software, los resultados intermedios y de manera más general, cualquier dato que sea de naturaleza transitoria. En forma más precisa, decimos que los datos de la base de datos "persisten" debido

en primer lugar a que una vez aceptados por el DBMS para entrar en la base de datos, *en lo sucesivo sólo pueden ser removidos de la base de datos por alguna solicitud explícita al DBMS,* no como un mero efecto lateral de (por ejemplo) algún programa que termina su ejecución. Por lo tanto, esta noción de persistencia nos permite dar una definición más precisa del término "base de datos":

■ Una **base de datos** es un conjunto de datos persistentes que es utilizado por los sistemas de aplicación de alguna empresa dada.

Aquí, el término "empresa" es simplemente un término genérico conveniente para identificar a cualquier organización independiente de tipo comercial, técnico, científico u otro. Una empresa podría ser un solo individuo (con una pequeña base de datos personal), toda una corporación o un gran consorcio similar (con una gran base de datos compartida) o todo lo que se ubique entre estas dos opciones. Aquí tenemos algunos ejemplos:

1. Una compañía manufacturera
2. Un banco
3. Un hospital
4. Una universidad
5. Un departamento gubernamental

Toda empresa necesariamente debe mantener una gran cantidad de datos acerca de su operación. Estos datos son los "datos persistentes" a los que nos referimos antes. En forma característica, las empresas que acabamos de mencionar incluirían entre sus datos persistentes a los siguientes:

1. Datos de producción
2. Datos contables
3. Datos de pacientes
4. Datos de estudiantes
5. Datos de planeación

Nota: Las primeras ediciones de este libro utilizaban el término "datos operacionales" en lugar de "datos persistentes". El primer término reflejaba el énfasis original en los sistemas de base de datos con aplicaciones **operacionales** o **de producción;** es decir, aplicaciones rutinarias altamente repetitivas que eran ejecutadas una y otra vez para apoyar la operación cotidiana de la empresa (por ejemplo, una aplicación para manejar los depósitos o retiros de efectivo en un sistema bancario). Para describir este tipo de entorno, se ha llegado a utilizar el término **procesamiento de transacciones en línea.** Sin embargo, ahora las bases de datos se utilizan cada vez más también para otro tipo de aplicaciones (por ejemplo, aplicaciones de **apoyo a la toma de decisiones)** y el término "datos operacionales" ya no es del todo apropiado. De hecho, hoy en días las empresas mantienen generalmente dos bases de datos independientes; una que contiene los datos operacionales y otra, a la que con frecuencia se le llama *almacén de datos (data warehouse),* que contiene datos de apoyo para la toma de decisiones. A menudo el almacén de datos incluye *información de resumen* (por ejemplo, totales, promedios...) y dicha información a su vez se extrae periódicamente de la base de datos operacional; digamos una vez al día o una vez por semana. Para una explicación más amplia de las bases de datos y las aplicaciones de apoyo a la toma de decisiones, consulte el capítulo 21.

Entidades y vínculos

Ahora consideraremos el ejemplo de una compañía manufacturera ("KnowWare Inc.") con un poco más de detalle. Por lo general, una empresa así desea registrar la información sobre los *proyectos* que maneja, las *partes* que utiliza en dichos proyectos, los *proveedores* que suministran esas partes, los *almacenes* en donde guardan esas partes, los *empleados* que trabajan en esos proyectos, etcétera. Por lo tanto los proyectos, partes, proveedores, etcétera, constituyen las **entidades** básicas de información que KnowWare Inc. necesita registrar (el término "entidad" es empleado comúnmente en los círculos de bases de datos para referirse a cualquier objeto distinguible que va a ser representado en la base de datos). Vea la figura 1.6.

Además de las propias entidades básicas (como los proveedores, las partes, etcétera, en el ejemplo), habrá también **vínculos** que asocian dichas entidades básicas. Estos vínculos están representados por los rombos y las líneas de conexión de la figura 1.6. Por ejemplo, existe un vínculo ("VP") entre proveedores y partes: cada proveedor suministra ciertas partes y de manera inversa, cada parte es suministrada por ciertos proveedores (para ser más precisos, cada proveedor suministra ciertos *tipos* de partes y cada *tipo* de parte es suministrado por ciertos proveedores). En forma similar, las partes son utilizadas en proyectos y de manera inversa, los proyectos utilizan partes (vínculo PY); las partes son guardadas en almacenes y los almacenes guardan partes (vínculo AP); y así sucesivamente. Observe que todos estos vínculos son *bidireccionales;* es decir, pueden ser recorridos en ambas direcciones. Por ejemplo, el vínculo VP entre proveedores y partes puede ser usado para responder las dos siguientes preguntas:

- Dado un proveedor, obtener las partes que éste suministra.
- Dada una parte, obtener los proveedores que la suministran.

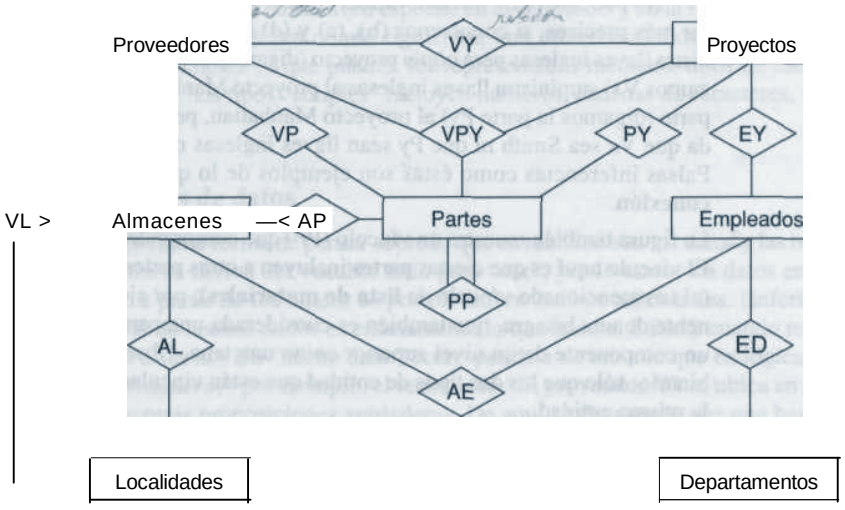


Figura 1.6 Diagrama de entidad/vínculo (E/R) para KnowWare Inc.

El punto importante con respecto a este vínculo (y por supuesto, con respecto a todos los vínculos de la figura) es que *son parte de los datos tanto como lo son las entidades básicas.* Por lo tanto, deben estar representados en la base de datos al igual que las entidades básicas. ***Nota:*** Como veremos en el capítulo 3, específicamente en un sistema relacional, tanto las **entidades** básicas como los vínculos que las conectan serán representados por medio de tablas como la que se muestra en la figura 1.1.

Observemos de paso que la figura 1.6 es un ejemplo de lo que se denomina (por razones obvias) **diagrama de entidad/vínculo** (diagrama E/R para abreviar). En el capítulo 13 consideraremos estos diagramas con un poco más de detalle.

La figura 1.6 ilustra además otros puntos importantes:

1. Aunque la mayoría de los vínculos de la figura comprenden *dos* tipos de entidad (es decir. son vínculos *binarios)* no significa que todos los vínculos deban ser necesariamente bina rios en este aspecto. En el ejemplo hay un vínculo ("VPY") que involucra tres tipos de en tidad (proveedores, partes y proyectos): un vínculo *ternario.* La interpretación que pretendo dar es que ciertos proveedores suministran ciertas partes para ciertos proyectos. Observe con cuidado que este vínculo ternario ("los proveedores suministran partes para proyectos") normalmente *no* equivale a la combinación de tres vínculos binarios "los proveedores sumi nistran partes", "las partes se usan en proyectos" y "los proyectos son abastecidos por los proveedores". Por ejemplo, la declaración de que

   (a) Smith suministra llaves inglesas para el proyecto Manhattan,

   nos dice *más* de lo que expresan las tres declaraciones siguientes:

   (b) Smith suministra llaves inglesas,

   (c) Las llaves inglesas se usan en el proyecto Manhattan y

   (d) El proyecto Manhattan es abastecido por Smith

   No podemos (¡de manera válida!) inferir (a) conociendo únicamente (b). (c) y (d). Para ser más precisos, si conocemos (b), (c) y (d) entonces podríamos inferir que Smith suministra llaves inglesas para *algún* proyecto (digamos el proyecto Yz), que *cierto* proveedor (digamos Vx) suministra llaves inglesas al proyecto Manhattan, y que Smith suministra *alguna* parte (digamos la parte Py) al proyecto Manhattan, pero no podemos inferir en forma válida que V*x* sea Smith ni que Py sean llaves inglesas ni que Yz sea el proyecto Manhattan. Falsas inferencias como éstas son ejemplos de lo que a veces se denomina **trampa de conexión.**

2. La figura también muestra un vínculo (PP) que comprende sólo *un* tipo de entidad (partes). El vínculo aquí es que ciertas partes incluyen a otras partes como componentes inmediatos (el tan mencionado vínculo de **lista de materiales);** por ejemplo, un tornillo es un compo nente de una bisagra, que también es considerada una parte y podría ser a su vez parte de un componente de un nivel superior como una tapa. Observe que el vínculo sigue siendo binario; sólo que los dos tipos de entidad que están vinculados (partes y partes) vienen a ser la misma entidad.

3. En general, un conjunto determinado de tipos de entidad podría vincularse entre sí en cualquier cantidad de vínculos distintos. En el ejemplo de la figura 1.6, hay dos vínculos distintos que involucran a proyectos y empleados: Uno (EY) representa el hecho de que los empleados están asignados a proyectos, el otro (MY) representa el hecho de que los em pleados administran proyectos.

Ahora observamos que *un vínculo puede considerarse como una entidad por derecho propio.* Si tomamos nuestra definición de entidad como "cualquier objeto acerca del cual queremos registrar información", entonces un vínculo se ajusta perfectamente a la definición. Por ejemplo, "la parte P4 está guardada en el almacén A8" es una entidad acerca de la cual bien querríamos registrar información (por ejemplo, la cantidad correspondiente). Más aún, podemos obtener ventajas definitivas (que exceden el alcance de este capítulo) no haciendo distinciones innecesarias entre entidades y vínculos. Por lo tanto, en este libro trataremos en su mayoría a los vínculos sólo como una clase especial de entidad.

Propiedades

Como acabamos de señalar, una entidad es cualquier objeto acerca del cual queremos registrar información. De donde se desprende que las entidades (incluidos los vínculos) poseen **propiedades** que corresponden a la información que deseamos registrar sobre ellas. Por ejemplo, los proveedores tienen *localidades;* las partes tienen *pesos;* los proyectos tienen *prioridades;* las asignaciones (de empleados a proyectos) tienen *fechas de inicio,* etcétera. Por lo tanto, dichas propiedades deben estar representadas en la base de datos. Por ejemplo, la base de datos podría incluir una tabla denominada V que represente a los proveedores y esa tabla podría incluir una columna de nombre CIUDAD que represente a las localidades de los proveedores.

En general, las propiedades pueden ser tan simples o tan complejas como queramos. Por ejemplo, la propiedad "localidad del proveedor" es supuestamente bastante simple, ya que sólo consiste en un nombre de ciudad y puede ser representada en la base de datos por una simple cadena de caracteres. En contraste, un almacén podría tener una propiedad "plan de piso", que podría ser bastante compleja, consistir tal vez en todo un dibujo arquitectónico y en el texto descriptivo asociado. Al momento de la publicación de este libro, la mayoría de los productos de bases de datos estaban apenas logrando manejar propiedades complejas como el dibujo y el texto. Regresaremos a este tema más delante (en especial en el capítulo 5 y en la Parte VI); mientras tanto, en la mayoría de los casos (en donde signifique una diferencia) daremos por hecho que las propiedades son "simples" y que pueden ser representadas mediante tipos de datos "simples". Los ejemplos de dichos tipos "simples" incluyen números, cadenas de caracteres, fechas, horas, etcétera.

Datos y modelos de datos

Existe otra importante forma de pensar sobre lo que en realidad son los datos y las bases de datos. La palabra *datos* se deriva del vocablo latín para "dar"; por lo tanto, los datos en realidad son *hechos dados,* a partir de los cuales es posible inferir hechos adicionales. (Inferir hechos adicionales a partir de hechos dados es exactamente lo que hace el DBMS cuando responde a una consulta de un usuario.) Un "hecho dado" corresponde a su vez a lo que en lógica se denomina *proposición verdadera;** por ejemplo, el enunciado "El proveedor V1 se ubica en Londres" podría ser una de estas proposiciones verdaderas. De aquí se desprende que una base de datos es en realidad *una colección de tales proposiciones verdaderas* [1.2].

---

*En lógica, una proposición es algo que se evalúa ya sea como *verdadero* o como *falso* en forma inequívoca. Por ejemplo, "William Shakespeare escribió *Pride and Prejudice"* es una proposición (por cierto, una falsa).

Una razón por la que los sistemas de bases de datos relaciónales se han vuelto tan dominantes, tanto en el mundo industrial como en el académico, es que manejan en forma muy directa (de hecho casi trivial) la interpretación precedente de los datos y las bases de datos. Los sistemas relaciónales están basados en una teoría formal denominada **el modelo de datos relacional,** de acuerdo con el la cual:

- En tablas, los datos son representados por medio de filas y estas filas pueden interpretarse directamente como proposiciones verdaderas. Por ejemplo, en la figura 1.1, la fila NICHO# 72 puede interpretarse en forma obvia como la siguiente proposición verdadera:

  "El nicho número 72 contiene dos botellas de Zinfandel Rafanelli 1995, y estarán listas para su consumo en el año 2003."

- Se proporcionan operadores para operar sobre las columnas de las tablas, y estos operadores soportan directamente el proceso de inferir proposiciones verdaderas adicionales a partir de las ya dadas. Como ejemplo sencillo, el operador relacional *proyectar* (vea la sección 1.6) nos permite inferir, a partir de la proposición verdadera que acabamos de citar, la siguiente proposición verdadera, entre otras:

  "Algunas botellas de Zinfandel estarán listas para su consumo en el año 2003."

  (para ser más precisos: "Algunas botellas de Zinfandel, en algún nicho, producidas por algún productor en algún año, estarán listas para su consumo en el año 2003.")

Sin embargo, el modelo relacional no es el único modelo de datos. Existen otros (vea la sección 1.6), aunque la mayoría de ellos difieren del modelo relacional en que son hasta cierto grado *específicos,* en vez de estar basados firmemente en la lógica formal. Sea lo que fuere, surge la pregunta: ¿En general qué *es* un modelo de datos? Podemos definir el concepto como sigue:

> Un **modelo de datos** es una definición lógica, independiente y abstracta de los objetos, operadores y demás que en conjunto constituyen la *máquina abstracta* con la que interactúan los usuarios. Los objetos nos permiten modelar la *estructura* de los datos. Los operadores nos permiten modelar su *comportamiento.*

Entonces, de manera útil, podemos distinguir al modelo de su *implementacion:*

- La **implementación** de determinado modelo de datos es una realización física, en una máqui na real, de los componentes de la máquina abstracta que en conjunto constituyen ese modelo.

Resumiendo: El modelo es aquello que los usuarios tienen que conocer; la implementación es lo que los usuarios no tienen que conocer.

Nota: Como puede ver, la distinción entre modelo e implementación es en realidad sólo un caso especial (uno muy importante) de la conocida distinción entre *lógico y físico.* Sin embargo, por desgracia muchos de los sistemas de bases de datos actuales (incluso aquellos que dicen ser relaciónales) no hacen estas distinciones con tanta claridad como debieran. De hecho, parece no haber un buen entendimiento de estas distinciones y de la importancia de hacerlas. Como consecuencia, a menudo hay una brecha entre los *principios* de las bases de datos (la forma en que los sistemas de bases de datos deberían funcionar) y la *práctica* de las bases de datos (la forma en que realmente funcionan). En este libro nos enfocamos principalmente en los principios; aunque es justo advertirle que cuando comience a utilizar un producto comercial, podría llevarse algunas sorpresas desagradables.

Para concluir esta sección, debemos mencionar el hecho de que en realidad el término *modelo de datos* es utilizado en la literatura con dos significados muy distintos. El primero es como se describió anteriormente. El segundo es como un modelo de los datos persistentes de alguna *empresa en particular* (por ejemplo, la compañía manufacturera KnowWare Inc. que mencionamos anteriormente en esta sección). La diferencia entre ambos significados puede ser caracterizada como sigue:

- En el primer sentido, un modelo de datos es como un *lenguaje de programación* (aunque en cierto modo abstracto) cuyos elementos pueden ser usados para resolver una amplia va riedad de problemas específicos, pero que en sí y por sí mismos no tienen una conexión di recta con ninguno de estos problemas específicos.
- En el segundo sentido, un modelo de datos es como un *programa específico* escrito en ese lenguaje. En otras palabras, un modelo de datos que toma las características que ofrece al gún modelo como el primero y las aplica a cierto problema específico. Puede ser visto como una *aplicación específica* de algún modelo con el primer significado.

De aquí en adelante, en este libro usaremos el término *modelo de datos* sólo en el primer sentido, excepto cuando se indique lo contrario.

1.4 ¿POR QUÉ UNA BASE DE DATOS?

¿Por qué utilizar un sistema de base de datos? ¿Cuáles son las ventajas? Hasta cierto punto, la respuesta a estas preguntas depende de si el sistema en cuestión es de un solo usuario o multiusuario (o para ser más precisos, existen muchas ventajas *adicionales* en el caso del sistema multiusuario). Consideraremos primero el caso de un solo usuario.

Vuelva a ver el ejemplo de la cava de vinos de la figura 1.1, el cual puede ser ilustrativo para el caso de un solo usuario. Ahora bien, esa base de datos es tan reducida y sencilla que las ventajas podrían no ser tan obvias. Pero imagine una base de datos similar para un gran restaurante, con una existencia tal vez de miles de botellas y cambios muy frecuentes a dichas existencias; o piense en una tienda de licores, también con una gran existencia y una mayor rotación en la misma. Tal vez en estos casos sea más fácil apreciar las ventajas de un sistema de base de datos sobre los métodos tradicionales basados en papel, para llevar un registro. He aquí algunas:

- *Compactación:* No hay necesidad de archivos en papel voluminosos.
- *Velocidad:* La máquina puede recuperar y actualizar datos más rápidamente que un humano. En particular, las consultas *específicas* sin mucha elaboración (por ejemplo, "¿Tenemos más Zinfandel que Pinot Noir?") pueden ser respondidas con rapidez, sin necesidad de búsquedas manuales o visuales que llevan tiempo.
- *Menos trabajo laborioso;* Se puede eliminar gran parte del trabajo de llevar los archivos a mano. Las tareas mecánicas siempre las realizan mejor las máquinas.
- *Actualidad:* En el momento que la necesitemos, tendremos a nuestra disposición información precisa y actualizada.

Desde luego, los beneficios anteriores se aplican aún con más fuerza en un entorno multiusuario, donde es probable que la base de datos sea mucho más grande y compleja que en el caso

de un solo usuario. No obstante, en el entorno multiusuario hay una ventaja adicional, que expresaremos así: *El sistema de base de datos ofrece a la empresa un control centralizado de sus datos* (los cuales, como se habrá dado cuenta a estas alturas, constituyen uno de sus activos más valiosos). Esta situación contrasta en gran medida con la que se encuentra en una empresa que no cuenta con un sistema de base de datos, en donde por lo general cada aplicación tiene sus propios archivos privados (a menudo también sus propias cintas y discos) de modo que los datos están muy dispersos y son difíciles de controlar de una forma sistemática.

Administración de datos y administración de bases de datos

Explicaremos brevemente este concepto del control centralizado. El concepto implica que en la empresa habrá alguna persona identificable que tendrá esta responsabilidad central sobre los datos. Esa persona es el **administrador de datos** (o DA) que mencionamos brevemente al final de la sección 1.2. Ya que (repitiendo) los datos son uno de los activos más valiosos de la empresa, es imperativo que exista una persona que los entienda junto con las necesidades de la empresa con respecto a esos datos, *a un nivel de administración superior.* Esa persona es el administrador de datos. Por lo tanto, es labor del administrador de datos decidir en primer lugar qué datos deben ser almacenados en la base de datos y establecer políticas para mantener y manejar esos datos una vez almacenados. Un ejemplo de estas políticas podría ser una que indicara quién puede realizar qué operaciones sobre ciertos datos y bajo qué circunstancias. En otras palabras, una política de *seguridad de los datos* (vea la siguiente subsección).

Hay que destacar que el administrador de datos es un administrador, no un técnico (aunque es cierto que necesita tener cierta idea de las posibilidades que tienen los sistemas de base de datos en el ámbito técnico). El *técnico* responsable de implementar las decisiones del administrador de datos es el **administrador de** *base* **de datos** (o DBA). Por lo tanto, el DBA, a diferencia del administrador de datos, es un *profesional IT.* El trabajo del DBA consiste en crear la base de datos real e implementar los controles técnicos necesarios para hacer cumplir las diversas decisiones de las políticas hechas por el administrador de datos. El DBA también es responsable de asegurar que el sistema opere con el rendimiento adecuado y de proporcionar una variedad de otros servicios técnicos. Por lo regular, el DBA tendrá un equipo de programadores de sistemas y otros asistentes técnicos (es decir, en la práctica la función del DBA normalmente es realizada por un equipo de personas, no por una sola); sin embargo, para fines de simplicidad, es conveniente suponer que el DBA es de hecho un solo individuo. En el capítulo 2 abordaremos con más detalle la función del DBA.

Beneficios del enfoque de base de datos

En esta subsección identificaremos algunas de las ventajas específicas que surgen de la noción anterior de control centralizado.

- *Los datos pueden compartirse*

  Explicamos este punto en la sección 1.2, pero para complementar lo mencionaremos de nuevo aquí. Compartir no sólo significa que las aplicaciones existentes puedan compartir la información de la base de datos, sino también que sea posible desarrollar nuevas aplicaciones para operar sobre los mismos datos. En otras palabras, es posible satisfacer los

requerimientos de datos de aplicaciones nuevas sin tener que agregar información a la base de datos.

Es posible reducir la redundancia

En sistemas que no son de bases de datos, cada aplicación tiene sus propios archivos exclusivos. A menudo este hecho puede conducir a una redundancia considerable de los datos almacenados, con el consecuente desperdicio de espacio de almacenamiento. Por ejemplo, una aplicación de personal y una aplicación de registro de escolaridad podrían tener un archivo que tuviera información departamental de los empleados. Sin embargo, como sugerí en la sección 1.2, estos dos archivos pueden integrarse y eliminar así la redundancia, en tanto el administrador de datos esté consciente de los requerimientos de datos de ambas aplicaciones; es decir, en tanto la empresa tenga el control general necesario.

Por cierto, no pretendemos sugerir que *toda* la redundancia pueda o deba necesariamente ser eliminada. En ocasiones hay razones sólidas, tácticas o del negocio, para mantener varias copias distintas de los mismos datos. Sin embargo, sí pretendemos sugerir que cualquier redundancia de este tipo debe ser **controlada** cuidadosamente; es decir, el DBMS debe estar al tanto de ella, si es que existe, y debe asumir la responsabilidad de "propagar las actualizaciones" (vea el siguiente punto).

Es posible (hasta cierto grado) evitar la inconsistencia

Éste es en realidad el corolario del punto anterior. Suponga que un hecho más real —digamos que el empleado E3 trabaja en el departamento D8— está representado por dos entidades distintas en la base de datos. Suponga también que el DBMS no está al tanto de esta duplicidad (es decir, la redundancia no está controlada). Entonces necesariamente habrá ocasiones en las que las dos entidades no coincidan: digamos, cuando una de ellas ha sido actualizada y la otra no. En esos momentos, decimos que la base de datos es *inconsistente.* Resulta claro que una base de datos en un estado inconsistente es capaz de proporcionar a sus usuarios información incorrecta o contradictoria.

Por supuesto, si el hecho anterior es representado por una sola entrada (es decir, si se elimina la redundancia), entonces no puede ocurrir tal inconsistencia. Como alternativa; si no se elimina la redundancia pero se *controla* (haciéndola del conocimiento del DBMS), entonces el DBMS puede garantizar que la base de datos nunca será inconsistente *a los ojos del usuario,* asegurando que todo cambio realizado a cualquiera de las dos entidades será aplicado también a la otra en forma automática. A este proceso se le conoce como **propagación de actualizaciones.**

Es posible brindar un manejo de transacciones

Una **transacción** es una unidad de trabajo lógica, que por lo regular comprende varias operaciones de la base de datos (en particular, varias operaciones de actualización). El ejemplo común es el de transferir una cantidad de efectivo de una cuenta A a otra cuenta *B.* Es claro que aquí se necesitan dos actualizaciones, una para retirar el efectivo de la cuenta A y la otra para depositarlo en la cuenta *B.* Si el usuario declara que las dos actualizaciones son parte de la misma transacción, entonces el sistema puede en efecto garantizar que se hagan ya sea ambas o ninguna de ellas, aun cuando el sistema fallara (digamos por falta de suministro eléctrico) a la mitad del proceso.

Nota: La característica de *atomicidad* de las transacciones que acabamos de ilustrar no es el único beneficio del manejo de transacciones, pero a diferencia de las otras característi-

cas, ésta se aplica aun en el caso de un solo usuario.* En los capítulos 14 y 15 aparece una descripción completa de las diversas ventajas del manejo de transacciones y de cómo pueden ser logradas.

Es posible mantener la integridad

El problema de la integridad es el de asegurar que los datos de la base de datos estén correctos. La inconsistencia entre dos entradas que pretenden representar el mismo "hecho" es un ejemplo de la falta de integridad (vea antes la explicación de este punto, en esta subsección); desde luego, este problema en particular puede surgir sólo si existe redundancia en los datos almacenados. No obstante, aun cuando no exista redundancia, la base de datos podría seguir conteniendo información incorrecta. Por ejemplo, un empleado podría aparecer con 400 horas laboradas durante la semana, en lugar de 40; o como parte de un departamento que no existe. El control centralizado de la base de datos puede ayudar a evitar estos problemas (en la medida de lo posible) permitiendo que el administrador de datos defina y el DBA implemente las **restricciones de integridad** (también conocidas como *reglas del negocio)* que serán verificadas siempre que se realice una operación de actualización.

Vale la pena señalar que la integridad de los datos es aún más importante en un sistema de base de datos que en un entorno de "archivos privados", precisamente porque los datos son compartidos. Sin los controles apropiados sería posible que un usuario actualizara la base de datos en forma incorrecta, generando así datos malos e "infectando" a otros usuarios con esos datos. También debemos mencionar que actualmente la mayoría de los productos de bases de datos son mas bien débiles con respecto al manejo de las restricciones de integridad (aunque ha habido algunas mejoras recientes en esta área). Éste es un hecho desafortunado, ya que (como veremos en el capítulo 8) las restricciones de integridad son fundamentales y de crucial importancia, mucho más de lo que por lo regular apreciamos.

Es posible hacer cumplir la seguridad

Al tener la completa jurisdicción sobre la base de datos, el DBA (por supuesto, bajo la dirección apropiada del administrador de datos) puede asegurar que el único medio de acceso a la base de datos sea a través de los canales adecuados y por lo tanto puede definir las reglas o **restricciones de seguridad** que serán verificadas siempre que se intente acceder a datos sensibles. Es posible establecer diferentes restricciones para cada tipo de acceso (recuperación, inserción, eliminación, etcétera) para cada parte de la información de la base de datos. Sin embargo, observe que sin dichas restricciones la seguridad de los datos podría de hecho estar en mayor riesgo que en un sistema de archivos tradicionales (dispersos); es decir, la naturaleza centralizada de un sistema de base de datos *requiere,* en cierto sentido, que también sea establecido un buen sistema de seguridad.

Es posible equilibrar los requerimientos en conflicto

Al conocer los requerimientos generales de la empresa (a diferencia de los requerimientos de los usuarios individuales), el DBA puede estructurar los sistemas de manera que ofrezcan un servicio general que sea "el mejor para la empresa" (de nuevo bajo la dirección del administrador de datos). Por ejemplo, es posible elegir una representación física de los datos

* Por otra parte, los sistemas de un solo usuario a menudo no proporcionan ningún tipo de manejo de transacciones sino que simplemente dejan el problema al usuario.

almacenados que proporcione un acceso rápido para las aplicaciones más importantes (posiblemente a costa de un acceso más lento para otras aplicaciones).

Es posible hacer cumplir los estándares

Con el control central de la base de datos, el DBA (una vez más, bajo la dirección del administrador de datos) puede asegurar que todos los estándares aplicables en la representación de los datos sean observados. Estos estándares podrían incluir alguno o todos los siguientes: departamentales, de instalación, corporativos, de la industria, nacionales e internacionales. Es conveniente estandarizar la representación de datos, en particular como un auxiliar para el *intercambio de datos* o para el movimiento de datos entre sistemas (esta consideración se ha vuelto particularmente importante con el advenimiento de los sistemas distribuidos; vea los capítulos 2 y 20). En forma similar, los estándares en la asignación de nombres y en la documentación de los datos también son muy convenientes como una ayuda para compartir y entender los datos.

Es probable que la mayoría de las ventajas mencionadas arriba sean bastante obvias. Sin embargo, es necesario agregar a la lista un punto que podría no ser tan obvio (aunque de hecho está implícito en otros); se trata de *dar independencia a los datos.* (Estrictamente hablando, éste es un *objetivo* de los sistemas de bases de datos, en vez de una ventaja). El concepto de la independencia de los datos es tan importante que le dedicamos una sección aparte.

1.5 LA INDEPENDENCIA DE LOS DATOS

Comenzaremos por observar que existen dos clases de independencia de los datos, física y lógica [1.3-1.4]; sin embargo, por el momento nos concentraremos sólo en la clase física. Por lo tanto, mientras no se diga otra cosa, el término no calificado "independencia de datos" deberá entenderse específicamente como independencia *física* de los datos. En los capítulos 2, 3 y en especial en el 9, abordaremos la independencia lógica de los datos. *Nota:* Tal vez también debamos decir que el término "independencia de los datos" no es muy adecuado (no capta muy bien la naturaleza de lo que en realidad está sucediendo); sin embargo, es el término utilizado tradicionalmente y nos apegaremos a él en este libro.

Podemos entender más fácilmente la independencia de los datos considerando a su opuesto. Las aplicaciones implementadas en sistemas más antiguos (los sistemas anteriores a los relacionales o incluso anteriores a las bases de datos) tienden a ser *dependientes de los datos. Esto* significa que la forma en que físicamente son representados los datos en el almacenamiento secundario y la técnica empleada para su acceso, son dictadas por los requerimientos de la aplicación en consideración, y más aún, significa que *el conocimiento de esa representación física y esa técnica de acceso están integrados dentro del código de la aplicación.*

- *Ejemplo:* Suponga que tenemos una aplicación que utiliza el archivo EMPLEADO de la figura 1.5 y suponga que se decidió, por motivos de rendimiento, que el archivo estaría indexado en su campo "nombre del empleado". En un sistema antiguo, la aplicación en cuestión generalmente estaría al tanto del hecho de que existe el índice, así como de la secuencia de registros que define ese índice; y la estructura de la aplicación estaría construida alrededor de ese conocimiento. En particular, la forma exacta de los diversos accesos a datos y rutinas de verificación de excepciones dentro de la aplicación, dependerá en gran

medida de los detalles de la interfaz que el software de administración de datos presenta a la aplicación.

Decimos que una aplicación como la del ejemplo es **dependiente de los datos,** debido a que es imposible modificar la representación física (la forma en que los datos están físicamente representados en el almacenamiento) o la técnica de acceso (la forma en que son accedidos físicamente) sin afectar a la aplicación de manera drástica. Por ejemplo, no sería posible reemplazar el índice del ejemplo por un esquema de dispersión sin hacer modificaciones mayores a la aplicación. Lo que es más, las partes de la aplicación que requieren de alteración en dicha situación son precisamente las partes que se comunican con el software de administración de datos; las dificultades implicadas son irrelevantes para el problema que originalmente debería resolver la aplicación; es decir, son dificultades *presentadas* por la naturaleza de la interfaz de administración de datos.

Sin embargo, en un sistema de base de datos, sería en extremo inconveniente permitir que las aplicaciones fuesen dependientes de los datos en el sentido descrito; por lo menos por las dos razones siguientes:

1. Las distintas aplicaciones requerirán visiones diferentes de los mismos datos. Por ejemplo, suponga que antes de que la empresa introduzca su base de datos integrada hay dos aplica ciones A y B que poseen cada una un archivo privado con el campo "saldo del cliente". Sin embargo, suponga que la aplicación A almacena dicho campo en formato decimal, mien tras que la aplicación B lo almacena en binario. Aún sería posible integrar los dos archivos y eliminar la redundancia, suponiendo que el DBMS esté listo y sea capaz de realizar todas las conversiones necesarias entre la representación almacenada elegida (la cual podría ser decimal o binaria, o quizá alguna otra) y la forma en que la aplicación desea verla. Por ejem plo, si se decide almacenar el campo en decimal, entonces todo acceso por parte de B re querirá una conversión hacia o desde el formato binario.

   Éste es un ejemplo muy trivial del tipo de diferencias que podrían existir en un sistema de base de datos entre los datos como los ve una aplicación dada y los datos como están almacenados físicamente. Más adelante en esta sección, consideraremos muchas otras posibles diferencias.

2. El DBA debe tener la libertad de cambiar las representaciones físicas o la técnica de acceso en respuesta a los requerimientos cambiantes, sin tener que modificar las aplicaciones exis tentes. Por ejemplo, es posible incorporar nuevos tipos de datos a la base de datos, adoptar nuevos estándares, cambiar las prioridades (y por lo tanto los requerimientos de rendimiento relativo), tener nuevos dispositivos disponibles, etcétera. Si las aplicaciones son dependien tes de los datos, estos cambios necesitarán por lo regular cambios correspondientes en los programas, ocupando así un esfuerzo de programación que de otro modo estaría disponible para la creación de nuevas aplicaciones. Aun en la actualidad, es muy común encontrar que una parte importante del esfuerzo de programación disponible, está dedicada a este tipo de mantenimiento (¡imagine el "problema del año 2000"!), lo cual por supuesto no es el mejor uso de un recurso escaso y valioso.

De aquí que dar independencia a los datos sea un objetivo principal de los sistemas de base de datos. Podemos definir la independencia de los datos como **la inmunidad de las aplicaciones a cambios en la representación física y en la técnica de acceso;** lo que implica desde luego que las aplicaciones involucradas no dependan de ninguna representación física o técnica de acceso

en particular. En el capítulo 2 describimos la arquitectura de los sistemas de bases de datos que proporciona el fundamento para lograr este objetivo. Sin embargo, antes de eso consideremos con mayor detalle algunos ejemplos de los tipos de cambios que el DBA desearía hacer y ante los cuales, por lo tanto, quisiéramos que las aplicaciones fuesen inmunes.

Comenzaremos por definir tres términos: *campo almacenado, registro almacenado* y *archivo almacenado* (consulte la figura 1.7).

Un **campo almacenado** es, en general, la unidad más pequeña de datos almacenados. La base de datos contendrá muchas **ocurrencias (o ejemplares)** de los diversos **tipos** de campos almacenados. Por ejemplo, una base de datos que contiene información sobre los diferentes tipos de partes podría incluir un tipo de campo almacenado con el nombre "número de parte" y luego podría existir una ocurrencia de ese campo almacenado para cada tipo de parte (tornillo, bisagra, tapa, etcétera).

Nota: En la práctica es común omitir los calificadores "tipo" y "ocurrencia" y depender del contexto para indicar a cuál se hace referencia. Aunque existe un ligero riesgo de

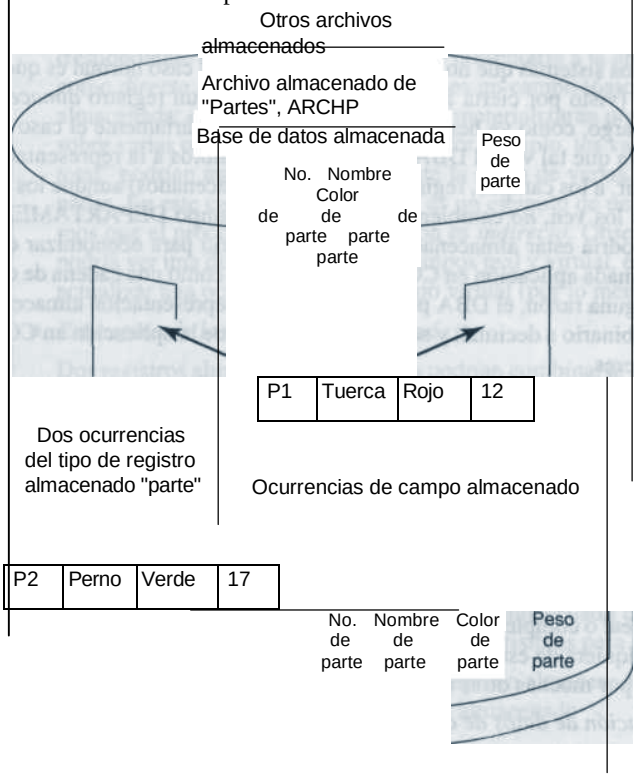


Figura 1.7 Archivos, registros y campos almacenados.

confusión, esta práctica es conveniente y nosotros mismos la adoptaremos de vez en cuando en el libro. (Esta observación se aplica también a los registros almacenados. Vea el siguiente párrafo.)

- Un **registro almacenado** es un conjunto de campos almacenados relacionados. Una vez más distinguimos entre tipo y ocurrencia. Una **ocurrencia (o ejemplar)** de registro almacenado consta de un grupo de ocurrencias de campos almacenados relacionados. Por ejemplo, una ocurrencia de registro almacenado dentro de la base de datos "partes" podría consistir en una ocurrencia de cada uno de los siguientes campos almacenados: número de parte, nombre de parte, color de parte y peso de parte. Decimos que la base de datos contiene muchas ocurrencias del **tipo** de registro almacenado "parte" (una vez más, una ocurrencia por cada clase de parte).
- Por último, un **archivo almacenado** es la colección de todas las ocurrencias existentes actualmente para un tipo de registro almacenado. *Nota:* Por simplicidad damos por hecho que todo archivo almacenado contiene sólo un tipo de registro almacenado. Esta simplificación no afecta sustancialmente ninguna de las explicaciones subsecuentes.

Ahora, en los sistemas que no son de bases de datos, el caso normal es que cualquier registro *lógico* dado (visto por cierta aplicación) es idéntico a un registro *almacenado* correspondiente. Sin embargo, como ya hemos visto éste no es necesariamente el caso en un sistema de base de datos, ya que tal vez el DBA necesita hacer cambios a la representación almacenada de datos (es decir, a los campos, registros y archivos almacenados) aunque los datos, tal y como las aplicaciones los ven, *no* cambien. Por ejemplo, el campo DEPARTAMENTO del archivo EMPLEADO podría estar almacenado en formato binario para economizar espacio, mientras que una determinada aplicación en COBOL podría verlo como una cadena de caracteres. Y más adelante, por alguna razón, el DBA podría modificar la representación almacenada de ese campo, digamos de binario a decimal, y seguir permitiendo que la aplicación en COBOL lo viese en forma de caracteres.

Como mencioné anteriormente, una diferencia como ésta, que comprende una conversión del tipo de datos de cierto campo en cada acceso, es comparativamente menor. No obstante, generalmente la diferencia entre lo que la aplicación ve y lo que en realidad está almacenado podría ser muy considerable. Para ampliar esta observación, presentamos a continuación una lista de los aspectos de la representación almacenada que podrían estar sujetos a cambio. En cada caso. usted deberá considerar lo que el DBMS tendría que hacer para que las aplicaciones sean inmunes a dicho cambio (si es que siempre puede lograrse esa inmunidad).

- *Representación de datos numéricos*

  Un campo numérico podría estar almacenado en la forma aritmética interna (por ejemplo, decimal empacado) o como una cadena de caracteres. En ambas formas, el DBA debe elegir una base apropiada (por ejemplo, binaria o decimal), una escala (de punto fijo o flotante), un modo (real o complejo) y una precisión (el número de dígitos). Podría ser necesario modificar cualquiera de estos aspectos para mejorar el rendimiento, para apegarse a un nuevo estándar o por muchas otras razones.

- *Representación de datos de caracteres*

  Un campo de cadena de caracteres podría ser almacenado mediante cualquiera de los distintos conjuntos de caracteres codificados (por ejemplo, ASCII, EBCDIC o Unicode).

Unidades para datos numéricos

Las unidades en un campo numérico podrían cambiar (por ejemplo, de pulgadas a centímetros durante un proceso de conversión al sistema métrico decimal).

Codificación de los datos

En ciertas situaciones podría ser conveniente representar los datos almacenados por medio de valores codificados. Por ejemplo, el campo "color de parte", que la aplicación ve como una cadena de caracteres ("Rojo" o "Azul" o "Verde" ...), podría ser almacenado como un solo dígito decimal, interpretado de acuerdo con el esquema de codificación 1 = "Rojo", 2 = "Azul", etcétera.

Materialización de los datos

En la práctica, el campo *lógico* (como lo ve una aplicación) corresponde por lo regular a cierto campo almacenado específico; aunque, como ya hemos visto, podría haber diferencias en el tipo de datos, la codificación, etcétera. En tal caso, el proceso de materialización (es decir, la construcción de una ocurrencia del campo lógico a partir de la ocurrencia correspondiente del campo almacenado y presentarla a la aplicación) podría ser considerado como *directo.* Sin embargo, en ocasiones un campo lógico no tendrá una sola contraparte almacenada; en su lugar, sus valores se materializarán por medio de algún cálculo, tal vez sobre varias ocurrencias almacenadas. Por ejemplo, los valores del campo lógico "cantidad total" podrían materializarse mediante la suma de varias cantidades individuales almacenadas. En este caso, "cantidad total" es un ejemplo de un campo **virtual,** por lo que decimos que el proceso de materialización es *indirecto.* Observe, sin embargo, que el usuario podría ver una diferencia entre los campos real y virtual, en tanto que podría no ser posible actualizar una ocurrencia de un campo virtual (por lo menos no directamente).

Estructura de los registros almacenados

Dos registros almacenados existentes podrían combinarse en uno. Por ejemplo, los registros almacenados

no. de parte	color de parte

no. de parte	peso de parte

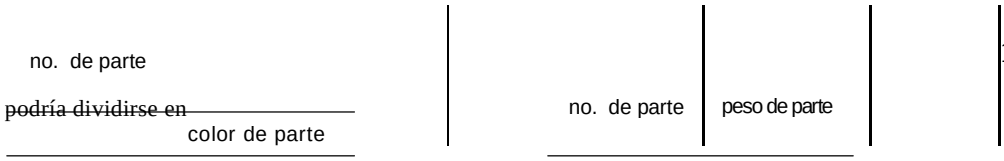
podrían combinarse para formar

no. de parte	color de parte	peso de parte

Un cambio así podría ocurrir cuando las aplicaciones existentes están integradas dentro del sistema de base de datos. Ello implica que el registro lógico de una aplicación podría consistir en un subconjunto propio del registro almacenado correspondiente; es decir, ciertos campos de ese registro almacenado serían invisibles para la aplicación en cuestión.

Como alternativa, un solo tipo de registro almacenado podría ser dividido en dos. Invirtiendo el ejemplo anterior, el registro almacenado

no. de parte	color de parte	peso de parte



1.6

Por ejemplo, esta separación permitiría que las porciones del registro original utilizadas con menos frecuencia sean almacenadas en un dispositivo más lento. Esto implica que un registro lógico de una aplicación podría contener campos de varios registros almacenados distintos; es decir, podría ser un superconjunto propio de cualquiera de esos registros almacenados.

■ *Estructura de los archivos almacenados*

Un determinado archivo almacenado puede ser implementado físicamente en el almacenamiento en una amplia variedad de formas. Por ejemplo, podría estar contenido completamente dentro de un solo volumen de almacenamiento (por ejemplo, un solo disco) o podría estar esparcido en varios volúmenes (posiblemente en diferentes tipos de dispositivos); podría o no tener una secuencia física de acuerdo con los valores de algún campo almacenado; podría o no tener otra secuencia de una o más formas en algún otro medio (digamos, en uno o más índices o en una o más cadenas de apuntadores insertados, o ambos); podría o no ser accesible mediante algún esquema de dispersión; los registros almacenados podrían o no estar bloqueados físicamente; y así sucesivamente. Pero ninguna de estas consideraciones deberá afectar de alguna manera a las aplicaciones (salvo, por supuesto, en el rendimiento).

Esto concluye nuestra lista de aspectos de la representación de datos almacenados que están sujetos a un posible cambio. La lista implica (entre otras cosas) que la base de datos podrá crecer sin dañar las aplicaciones existentes; de hecho, permitir que la base de datos crezca sin dañar de manera lógica las aplicaciones existentes es una de las razones más importantes para requerir, en primer lugar, la independencia de los datos. Por ejemplo, debe ser posible ampliar un registro almacenado existente agregando nuevos campos almacenados, lo que de manera típica representa más información con respecto a algún tipo de entidad existente (por ejemplo, un campo de "costo unitario" podría ser agregado al registro almacenado "parte"). Estos nuevos campos deben simplemente ser invisibles para las aplicaciones existentes. En forma similar, debe ser posible agregar tipos de registros almacenados completamente nuevos (y por lo tanto nuevos archivos almacenados), sin necesidad de algún cambio a las aplicaciones existentes; generalmente, tales registros representarían nuevos tipos de entidad (por ejemplo, un tipo de registro "proveedor" podría ser agregado a la base de datos "partes"). De nuevo, dichas adiciones deberán ser invisibles para las aplicaciones existentes.

A esta altura, quizás ya se ha dado cuenta de que la independencia de los datos es una de las razones de por qué es tan importante separar el modelo de datos de su implementación (como se explicó casi al final de la sección 1.3): En el grado en que no hagamos esa separación, no lograremos la independencia de los datos. Por lo tanto, la falla general al no realizar apropiadamente dicha separación, en especial en los sistemas SQL actuales, es particularmente angustiante. *Nota:* Con estas observaciones no pretendemos decir que los sistemas SQL actuales no proporcionen en absoluto la independencia de los datos, sólo queremos indicar que proporcionan mucho menos de lo que en teoría son capaces los sistemas relaciónales. En otras palabras, la independencia de los datos no es algo absoluto (diferentes sistemas proporcionan distintos grados de independencia y algunos, si los hay, no ofrecen ninguna); los sistemas SQL ofrecen más que los sistemas antiguos, pero aún no son perfectos, como veremos en los capítulos que vienen.

1.6 LOS SISTEMAS RELACIÓNALES Y OTROS SISTEMAS

Como mencioné al final de la sección 1.3, los productos DBMS que se basan en *el modelo de datos relacional* (los "sistemas relaciónales") han venido a dominar el mercado de las bases de datos. Lo que es más, la mayor parte de la investigación sobre bases de datos en los últimos 30 años, se ha basado (aunque en algunos casos un poco en forma indirecta) en este modelo. De hecho, la presentación del modelo relacional en 1969-70 fue de manera innegable el evento más importante en toda la historia de las bases de datos. Por estas razones —más la razón de que el modelo relacional está sólidamente fundamentado en la lógica y en las matemáticas y por lo tanto ofrece un vehículo ideal para la enseñanza de los principios de las bases de datos— el énfasis de este libro (como señalé en la sección 1.1) descansa en gran medida en los sistemas relaciónales.

¿Qué es exactamente un sistema relacional? Es obvio en este punto tan inicial del libro, que no es posible contestar esta pregunta por completo; pero es posible e incluso conveniente dar una respuesta ordinaria y pronta, la cual podremos detallar más adelante. Brevemente, y muy vagamente, un sistema relacional es aquél en el que:

'1. Los datos son percibidos por el usuario como tablas (y nada más que tablas); y

'2. Los operadores disponibles para el usuario (por ejemplo, para recuperación) son operadores que generan nuevas tablas a partir de las anteriores. Por ejemplo, hay un operador *restringir* que extrae un subconjunto de filas de una tabla dada y otro operador proyector que extrae un subconjunto de columnas; y por supuesto, un subconjunto de filas y un subconjunto de columnas de una tabla pueden de por sí ser vistos como tablas, como veremos en un momento.

Nota: La razón por la que dichos sistemas se denominan "relaciónales" es que el término *relación* es básicamente el término matemático para *tabla;* de hecho, los términos *relación* y *tabla* pueden tomarse como sinónimos, por lo menos para fines informales (para una mayor explicación, vea los capítulos 3 y 5). Quizá debamos agregar que la razón *no* es en definitiva que el término *relación* sea "básicamente un término matemático para" un *vínculo* en el sentido de los diagramas de entidad/vínculo (vea la sección 1.3); de hecho, existe muy poca conexión directa entre los sistemas relaciónales y dichos diagramas, como veremos en el capítulo 13.

Para repetir, posteriormente detallaremos las definiciones anteriores; pero éstas servirán por el momento. La figura 1.8 ofrece una ilustración. Los datos (ver la parte a. de la figura) consisten en una sola tabla, denominada CAVA (de hecho, ésta es una versión de la tabla CAVA de la figura 1.1, que fue reducida para hacerla más manejable). En la parte b. de la figura se muestran dos recuperaciones de ejemplo; una que comprende una *restricción* u operación del subconjunto de filas y la otra una *proyección* u operación del subconjunto de columnas. *Nota:* Una vez más, las dos recuperaciones están expresadas en SQL.

Ahora podemos distinguir entre los sistemas relaciónales y los no relaciónales de la siguiente manera. Como ya mencionamos, el usuario de un sistema relacional ve tablas y nada más que tablas. En contraste, el usuario de un sistema no relacional ve otras estructuras de datos, ya sea en lugar de las tablas de un sistema relacional o además de ellas. A su vez, esas otras estructuras requieren de otros operadores para manipularlas. Por ejemplo, en un sistema **jerárquico** como el IMS de IBM, los datos son representados ante el usuario como un conjunto de estructuras de árbol (jerarquías), y los operadores que se proporcionan para manipular dichas estructuras in-

a. *Tabla dada:* CAVA

VINO	AÑO	BOTELLAS
Chardonnay	1996	4
Fumé Blanc	1996	2
Pinot Noir	1993	3
Zinfandel	1994	9

b. *Operadores (ejemplos):*

1. *Restringir:* *Resultado:*

```
SELECT VINO, AÑO, BOTELLAS
FROM CAVA WHERE AÑO > 1995
;
```

VINO	AÑO	BOTELLAS
Chardonnay	1996	4
Fumé Blanc	1996	2

2. *Proyectar:* *Resultado:*

```
SELECT VINO, BOTELLAS
FROM  CAVA ;
```

VINO	BOTELLAS
Chardonnay	4
Fumé Blanc	2
Pinot Noir	3
Zinfandel	9

Figura 1.8 Estructura de datos y operadores en un sistema relacional (ejemplos).

cluyen operadores para *apuntadores de recorrido; es decir,* los apuntadores que representan las rutas jerárquicas hacia arriba y hacia abajo en los árboles. (En contraste, es una característica distintiva de los sistemas relaciónales el que no contengan, como hemos visto, dichos apuntadores.)

Para llevar este aspecto un poco más lejos: Los sistemas de bases de datos pueden de hecho ser divididos convenientemente en categorías de acuerdo con los operadores y estructuras de datos que presentan al usuario. De acuerdo con este esquema, los sistemas más antiguos (prerrelacionales) se ubican dentro de tres grandes categorías, los sistemas **de listas invertidas, jerárquicos** y **de red.*** En este libro no exponemos con detalle estas categorías debido a que, por lo menos desde un punto de vista tecnológico, deben ser vistos como obsoletos. (Si usted está interesado, puede encontrar descripciones tutoriales de los tres en la referencia [1.5].) Sin embargo, debemos mencionar por lo menos que el término *red* en este contexto no tiene nada que ver con una red de *comunicaciones;* más bien se refiere (para repetir) a las clases de estructuras de datos y operadores que manejan los sistemas en cuestión.

---

*Por analogía con el modelo relacional, las ediciones anteriores de este libro se referían a los *modelos* de listas invertidas, jerárquicos y de red (y gran parte de la literatura aún lo hace). Sin embargo, hablar en esos términos es más bien inexacto, ya que a diferencia del modelo relacional, los "modelos" de listas invertidas, jerárquicos y de red se definieron *después del hecho;* es decir, los productos comerciales de listas invertidas, jerárquicos y de red fueron implementados *primero* y los "modelos" correspondientes fueron definidos *posteriormente* mediante un proceso de inducción —en este contexto, un término formal para adivinar— a partir de las implementaciones existentes.

Nota: En ocasiones, a los sistemas de red se les denomina sistemas **CODASYL** o sistemas **DBTG,** por el grupo que los propuso; es decir, el DBTG (Grupo de Tareas de Bases de Datos) del CODASYL (Congreso sobre Lenguajes de Sistemas de Datos). Probablemente el ejemplo más conocido de estos sistemas es IDMS (de Computer Associates International Inc). Al igual que los sistemas jerárquicos, aunque a diferencia de los relaciónales, todos estos sistemas exponen (entre otras cosas) apuntadores ante el usuario.

Los primeros productos **relaciónales** comenzaron a aparecer a finales de los años setenta y principios de los ochenta. Hasta este momento, la gran mayoría de los sistemas de base de datos son relaciónales y operan prácticamente en todo tipo de plataforma de hardware y de software disponible. Los ejemplos principales comprenden, en orden alfabético, a DB2 (varias versiones) de IBM Corp.; Ingress II de Computer Associates International Inc.; Informix Dynamic Server de Informix Software Inc.; Microsoft SQL Server de Microsoft Corp.; Oracle 8i de Oracle Corp.; y Sybase Adaptive Server de Sybase Inc. *Nota:* Cuando tengamos motivos para referirnos a cualquiera de estos productos en el resto del libro, lo haremos por sus nombres abreviados(como lo hace la mayoría de la industria, de manera informal): DB2, Ingres, Informix, SQL Server, Oracle y Sybase, respectivamente.

En fechas más recientes, se han puesto a disposición ciertos productos de **objetos** y **objeto/reladonales.*** Los sistemas objeto/relaciónales representan en su mayoría extensiones compatibles hacia arriba de algunos de los productos relaciónales originales, como es el caso de DB2 e Informix; los sistemas de objetos (en ocasiones *orientados a objetos)* representan intentos de hacer algo completamente diferente, como es el caso de GemStone de GemStone Systems Inc. y de Versant ODBMS de Versant Object Technology. En la parte VI de este libro, explicaremos estos sistemas más recientes.

Además de los distintos enfoques antes mencionados, a lo largo de los años las investigaciones han seguido una variedad de esquemas alternativos, incluyendo los enfoques **multidimensional** y el **basado en la lógica** (también llamado *deductivo* o *experto).* En el capítulo 21 explicaremos los sistemas multidimensionales y en el capítulo 23 los sistemas basados en la lógica.

1.7 RESUMEN

Cerramos este capítulo introductorio con un resumen de las ideas principales expuestas. Primero, es posible pensar en un **sistema de base de datos** como un sistema de registros computarizado. Dicho sistema comprende a los propios **datos** (almacenados en la **base de datos),** al **hardware,** al **software** (en particular al **sistema de administración de base de datos** o DBMS) y (¡lo más importante!) a los **usuarios.** A su vez, los usuarios pueden ser divididos en **programadores de aplicaciones, usuarios finales** y **administrador de base de datos** o DBA. El DBA es el responsable de administrar la base de datos y el sistema de base de datos, de acuerdo con las políticas establecidas por el **administrador de datos.**

Las bases de datos están **integradas** y por lo regular son **compartidas;** se emplean para almacenar **datos persistentes.** Dichos datos pueden considerarse, de manera útil aunque informal, como una representación de **entidades,** junto con los **vínculos** que están entre éstas (aunque de hecho, un vínculo es en realidad sólo una clase especial de entidad). Analizaremos brevemente la idea de los **diagramas de entidad/vínculo.**

---

*Aquí el término *objeto* tiene un significado más bien específico, el cual explicaremos en la parte VI. Antes de ese punto, usaremos el término en su sentido general, salvo que se indique lo contrario.

Los sistemas de bases de datos ofrecen diversos beneficios. Uno de los más importantes es el de la **independencia** (física) **de los datos.** Podemos definir la independencia de los datos como la inmunidad que tienen los programas de aplicación ante los cambios en la forma almacenar o acceder físicamente a los datos. Entre otras cosas, la independencia de los datos requiere que se haga una clara distinción entre el **modelo de datos y** su **implementación.** (De paso, le recordamos que el término *modelo de datos,* quizás en forma desafortunada, tiene dos significados diferentes.)

Los sistemas de bases de datos también soportan por lo regular **transacciones** o unidades de trabajo lógicas. Una ventaja de las transacciones es que está garantizado que sean **atómicas** (todo o nada), incluso si el sistema falla a mitad de su ejecución.

Por último, los sistemas de bases de datos pueden estar fundamentados en varias teorías diferentes. En particular, los sistemas relaciónales se basan en una teoría formal denominada **modelo relacional,** según la cual los datos están representados como filas de tablas (interpretadas como **proposiciones verdaderas)** y cuentan con operadores que manejan directamente el proceso de **inferir** proposiciones verdaderas adicionales a partir de las ya dadas. Desde una perspectiva tanto económica como teórica, los sistemas relacionales son sin duda los más importantes (y no es probable que esta situación cambie en el futuro previsible). Vimos algunos ejemplos de SQL, el lenguaje estándar para tratar con los sistemas relacionales (en particular, ejemplos de las instrucciones **SELECT, INSERT, UPDATE y DELETE** de SQL). Este libro se basará en gran medida en los sistemas relacionales, aunque por las razones que expuse en el prefacio, no demasiado en SQL *per se.*

EJERCICIOS

1.1 Defina los siguientes términos:

acceso concurrente
administración de datos
aplicación en línea
archivo almacenado
base de datos
campo almacenado
compartir
datos persistentes
DBA
DBMS
diagrama de entidad/vínculo
entidad
independencia de los datos
integración
integridad
interfaz controlada por comandos
interfaz controlada por formularios
interfaz controlada por menus
lenguaje de consulta
propiedad
redundancia
registro almacenado
seguridad
sistema de base de datos
sistema multiusuario
transacción
vínculo
vínculo binario

1.2 ¿Cuáles son las ventajas de usar un sistema de base de datos?

1.3 ¿Cuáles son las desventajas de usar un sistema de base de datos?

1.4 ¿Qué entiende por el término *sistema relacionan* Distinga entre los sistemas relacionales y los no relacionales.

1.5 ¿Qué entiende por el término *modelo de datos?* Explique la diferencia entre un modelo de datos y su implementación. ¿Por qué es importante la diferencia?

1.6 Muestre los efectos que tienen las siguientes operaciones SQL de recuperación sobre la base de datos de la cava de vinos mostrada en la figura 1.1.

a.
```
SELECT VINO, PRODUCTOR
FROM   CAVA
WHERE NICHO# • 72 ;
```

b.
```
SELECT VINO, PRODUCTOR
FROM   CAVA
WHERE   AÑO > 1996  ;
```

C.
```
SELECT NICHO#,  VINO,  AÑO
FROM    CAVA WHERE    LISTO <
1999  ;
```

d.
```
SELECT VINO,  NICHO#,  AÑO
FROM    CAVA
WHERE   PRODUCTOR =  ' Robt.  Mondavi'
AND     BOTELLAS > 6  ;
```

1.7 A partir de cada una de sus respuestas al ejercicio 1.6, dé en sus propias palabras una interpre tación (como una proposición verdadera) de una fila típica.

1.8 Muestre los efectos de las siguientes operaciones SQL de actualización sobre la base de datos de la cava de vinos de la figura 1.1.

a.
```
INSERT
INTO  CAVA ( NICH0#, VINO, PRODUCTOR, AÑO, BOTELLAS, LISTO )
VALUES ( 80, 'Syrah', 'Meridian', 1994, 12, 1999 ) ;
```

b.
```
DELETE
FROM  CAVA
WHERE LISTO > 2000 ;
```

C.
```
UPDATE CAVA
SET   BOTELLAS = 5
WHERE NICHO# = 50 ;
```

d.
```
UPDATE CAVA
SET   BOTELLAS = BOTELLAS + 2
WHERE  NICHO# = 50 ;
```

1.9 Escriba instrucciones SQL para realizar las siguientes operaciones en la base de datos de la cava de vinos:

a. Obtenga el número de nicho, el nombre del vino y el número de botellas de todos los vinos Geyser Peak.
b. Obtenga el número de nicho y el nombre de todos los vinos que tengan en existencia más de cinco botellas.
c. Obtenga el número de nicho de todos los vinos rojos.
d. Agregue tres botellas al nicho número 30.
e. Elimine de las existencias todo el Chardonnay.
f. Agregue una entrada para un nuevo caso (12 botellas) de Gary Farrell Merlot: nicho número 55, año 1996, listo en el 2001.

1.10 Suponga que tiene una colección de música clásica que consta de CDs, LPs y cintas de audio y desea elaborar una base de datos que le permita determinar qué grabaciones posee de un compositor específico (por ejemplo, Sibelius), director (por ejemplo, Simon Rattle), solista (por ejemplo, Arthur Grumiaux), obra (por ejemplo, la quinta sinfonía de Beethoven), orquesta (por ejemplo, la Orquesta filarmónica de la ciudad de Nueva York), tipo de obra (por ejemplo, concierto para violin) o grupo de cámara (por ejemplo, Kronos Quartet). Dibuje un diagrama de entidad/vínculo para esta base de datos (como el de la figura 1.6).

REFERENCIAS Y BIBLIOGRAFÍA

1.1 E. F. Codd: "Data Models in Database Management", Proc. Wokshop on Data Abstraction, Databases, and Conceptual Modelling, Pingree Park, Colo, (junio, 1980); *ACM SIGART Newsletter No.* 74 (enero, 1981); *ACM SIGMODRecord 11*,No. 2(febrero, 1981);*ACM SIGPLAN Notices 16,No.l* (ene ro, 1981).

Codd fue el inventor del modelo relacional, el cual describió primero en la referencia [5.1]. Sin embargo, dicha referencia ¡no define el término *modelo de datos* como tal!; aunque el presente artículo (muy posterior), sí lo hace. Éste aborda la pregunta ¿A qué fines pretenden servir los modelos de datos en general y el modelo relacional en particular? Después continúa ofreciendo evidencia para apoyar la afirmación de que, al contrario de la creencia popular, el modelo relacional fue de hecho el primer modelo de datos en ser definido. (En otras palabras, Codd afirma en cierto modo ser el inventor del concepto de modelo de datos en general, así como del modelo de datos relacional en particular.)

1.2 Hugh Darwen: "What a Database *Really Is:* Predicates and Propositions", en C. J. Date, Hugh Darwen y David McGoveran, *Relational Database Writings 1994-1997.* Reading, Mass.: Addison-Wesley (1998).

Este artículo ofrece una explicación informal (pero precisa) de la idea, expuesta brevemente al final de la sección 1.3, de que es posible visualizar a una base de datos como un conjunto de proposiciones verdaderas.

1.3 C. J. Date y P. Hopewell: "Storage Structures and Physical Data Independence", Proc. 1971 ACM SIGFIDET Workshop on Data Definition, Access, and Control, San Diego, California (noviembre, 1971).

1.4 C. J. Date y P. Hopewell: "File Definition and Logical Data Independence", Proc. 1971 ACM SIG FIDET Workshop on Data Definition, Access, and Control, San Diego, California (noviembre, 1971).

Las referencias [1.3-1.4] fueron los primeros artículos en definir y distinguir entre la independencia física y lógica de los datos.

1.5 C. J. Date: *Relational Database Writings 1991-1994.* Reading, Mass.: Addison-Wesley (1995).

RESPUESTAS A EJERCICIOS SELECCIONADOS

1.3 Algunas desventajas son las siguientes:

- Podría comprometerse la seguridad (sin los controles adecuados);
- Podría comprometerse la integridad (sin los controles adecuados);
- Podría requerirse de hardware adicional;
- La sobrecarga en el rendimiento podría ser importante;