

The background of the cover is a photograph of several sailboats with large red sails on a body of water during a sunset or sunrise. The sky is filled with soft, colorful clouds in shades of orange, yellow, and blue. The water reflects the light from the sky and the sails. The silhouettes of the sailboats and the people on board are visible against the bright horizon.

# FUNDAMENTOS DE BASES DE DATOS

QUINTA EDICIÓN

**Mc  
Graw  
Hill**

SILBERSCHATZ | KORTH | SUDARSHAN



Además de utilizar la información que pueda tener el sistema respecto de la solicitud que se esté procesando, el gestor de la memoria intermedia puede utilizar información estadística relativa a la probabilidad de que una solicitud haga referencia a una relación concreta. Por ejemplo, el diccionario de datos (como se verá con detalle en el Apartado 11.8) que realiza un seguimiento del esquema lógico de las relaciones y de la información de su almacenamiento físico es una de las partes de la base de datos a la que se tiene acceso con mayor frecuencia. Por tanto, el gestor de la memoria intermedia debe intentar no eliminar de la memoria principal los bloques del diccionario de datos, a menos que se vea obligado a hacerlo. En el Capítulo 12 se estudian los índices de los archivos. Dado que puede que se acceda más frecuentemente al índice del archivo que al propio archivo, el gestor de la memoria intermedia no deberá, en general, eliminar de la memoria principal los bloques del índice si se dispone de alternativas.

La estrategia ideal para la sustitución de bloques necesita información sobre las operaciones de las bases de datos (las que se estén realizando y las que se vayan a realizar en el futuro). No se conoce una sola estrategia que sea adecuada para todas las situaciones posibles. En realidad, un número sorprendentemente grande de bases de datos utiliza LRU, a pesar de sus defectos. Las preguntas prácticas y los ejercicios exploran estrategias alternativas.

La estrategia utilizada por el gestor de la memoria intermedia para la sustitución de los bloques se ve influida por factores distintos del momento en que se volverá a hacer referencia a cada bloque. Si el sistema está procesando de manera concurrente las solicitudes de varios usuarios, puede que el subsistema para el control de concurrencia (Capítulo 16) tenga que posponer ciertas solicitudes para asegurar la conservación de la consistencia de la base de datos. Si se proporciona al gestor de la memoria intermedia información del subsistema de control de concurrencia que indique las solicitudes que se posponen, puede utilizar esa información para modificar su estrategia de sustitución de los bloques. Concretamente, los bloques que necesiten las solicitudes activas (no pospuestas) pueden conservarse en la memoria intermedia a expensas de los que necesiten las solicitudes pospuestas.

El subsistema para la recuperación de caídas (Capítulo 17) impone severas restricciones a la sustitución de bloques. Si se ha modificado un bloque, no se permite que el gestor de la memoria intermedia vuelva a copiar al disco la versión nueva del bloque existente en la memoria intermedia, dado que eso destruiría la versión anterior. Por el contrario, el gestor de bloques debe solicitar permiso del subsistema para la recuperación de caídas antes de escribir cada bloque. Puede que el subsistema para la recuperación de caídas exija que se fuerce la salida de otros bloques antes de conceder autorización al gestor de la memoria intermedia para que escriba el bloque solicitado. En el Capítulo 17 se define con precisión la interacción entre el gestor de la memoria intermedia y el subsistema para la recuperación de caídas.

## 11.6 Organización de los archivos

Los **archivos** se organizan lógicamente como secuencias de registros. Esos registros se corresponden con los bloques del disco. Los archivos constituyen un elemento fundamental de los sistemas operativos, por lo que se supone la existencia de un *sistema de archivos* subyacente. Hay que tomar en consideración diversas maneras de representar los modelos lógicos de datos en términos de los archivos.

Aunque los bloques son de un tamaño fijo determinado por las propiedades físicas del disco y por el sistema operativo, el tamaño de los registros varía. En las bases de datos relacionales, las tuplas de las diferentes relaciones suelen ser de tamaño diferente.

Un enfoque de la correspondencia entre base de datos y archivos es utilizar varios archivos y guardar los registros de la misma longitud en un mismo archivo. Una alternativa es estructurar los archivos de modo que puedan aceptar registros de longitudes diferentes; no obstante, los archivos con registros de longitud fija son más sencillos de implementar que los que tienen registros de longitud variable. Muchas de las técnicas empleadas para los primeros pueden aplicarse a los de longitud variable. Por tanto, se comienza por tomar en consideración los archivos con registros de longitud fija.

### 11.6.1 Registros de longitud fija

A manera de ejemplo, considérese un archivo con registros de *cuentas* de la base de datos bancaria. Cada registro de este archivo se define (en pseudocódigo) de la manera siguiente:

```

type depósito = record
    número_cuenta char(10);
    nombre_sucursal char(22);
    saldo numeric(12,2);
end

```

Si se supone que cada carácter ocupa un byte y que los valores de tipo numeric(12,2) ocupan ocho bytes, el registro de *cuenta* tiene cuarenta bytes de longitud. Un enfoque sencillo es utilizar los primeros cuarenta bytes para el primer registro, los cuarenta bytes siguientes para el segundo, y así sucesivamente (Figura 11.5). Sin embargo, hay dos problemas con este sencillo enfoque:

1. Resulta difícil borrar registros de esta estructura. Hay que rellenar el espacio ocupado por el registro que se va a borrar con algún otro registro del archivo, o tener alguna manera de marcar los registros borrados para poder pasarlos por alto.
2. A menos que el tamaño de los bloques sea múltiplo de cuarenta (lo que resulta improbable), algún registro se saltará los límites de los bloques. Es decir, parte del registro se guardará en un bloque y parte en otro. Harán falta, por tanto, dos accesos a bloques para leer o escribir esos registros.

Cuando se borra un registro, se puede desplazar el situado a continuación al espacio ocupado que ocupaba el registro borrado y hacer lo mismo con los demás, hasta que todos los registros situados a continuación del borrado se hayan desplazado hacia delante (Figura 11.6). Este tipo de enfoque necesita desplazar gran número de registros. Puede que fuera más sencillo desplazar simplemente el último registro del archivo al espacio ocupado por el registro borrado (Figura 11.7).

No resulta deseable desplazar los registros para que ocupen el espacio liberado por los registros borrados, ya que para hacerlo se necesitan accesos adicionales a los bloques. Dado que las operaciones de inserción tienden a ser más frecuentes que las de borrado, resulta aceptable dejar libre el espacio

registro 0	C-102	Navacerrada	400
registro 1	C-305	Collado Mediano	350
registro 2	C-215	Becerril	700
registro 3	C-101	Centro	500
registro 4	C-222	Moralzarzal	700
registro 5	C-201	Navacerrada	900
registro 6	C-217	Galapagar	750
registro 7	C-110	Centro	600
registro 8	C-218	Navacerrada	700

**Figura 11.5** Archivo que contiene los registros de *cuenta*.

registro 0	C-102	Navacerrada	400
registro 1	C-305	Collado Mediano	350
registro 3	C-101	Centro	500
registro 4	C-222	Moralzarzal	700
registro 5	C-201	Navacerrada	900
registro 6	C-217	Galapagar	750
registro 7	C-110	Centro	600
registro 8	C-218	Navacerrada	700

**Figura 11.6** El archivo de la Figura 11.5 con el registro 2 borrado y todos los registros desplazados.

registro 0	C-102	Navacerrada	400
registro 1	C-305	Collado Mediano	350
registro 8	C-218	Navacerrada	700
registro 3	C-101	Centro	500
registro 4	C-222	Moralzarzal	700
registro 5	C-201	Navacerrada	900
registro 6	C-217	Galapagar	750
registro 7	C-110	Centro	600

**Figura 11.7** El archivo de la Figura 11.5 con el registro 2 borrado y el último registro desplazado.

ocupado por los registros borrados y esperar a una inserción posterior antes de volver a utilizar ese espacio. No basta con una simple marca en el registro borrado, ya que resulta difícil hallar el espacio disponible mientras se realiza una inserción. Por tanto, hay que introducir una estructura adicional.

Al comienzo del archivo se asigna cierto número de bytes como **cabecera del archivo**. La cabecera contiene gran variedad de información sobre el archivo. Por ahora, todo lo que hace falta guardar es la dirección del primer registro cuyo contenido se haya borrado. Se utiliza este primer registro para guardar la dirección del segundo registro disponible, y así sucesivamente. De manera intuitiva se pueden considerar estas direcciones guardadas como *punteros*, dado que indican la posición de un registro. Los registros borrados, por tanto, forman una lista enlazada a la que se suele denominar **lista libre**. La Figura 11.8 muestra el archivo de la Figura 11.5, con la lista libre, después de haberse borrado los registros 1, 4 y 6.

Al insertar un registro nuevo se utiliza el registro al que apunta la cabecera. Se modifica el puntero de la cabecera para que señale al siguiente registro disponible. Si no hay espacio disponible, se añade el nuevo registro al final del archivo.

La inserción y el borrado de archivos con registros de longitud fija son sencillas de implementar, dado que el espacio que deja libre cada registro borrado es exactamente el mismo que se necesita para insertar otro registro. Si se permiten en un archivo registros de longitud variable, esta coincidencia no se mantiene. Puede que el registro insertado no quepa en el espacio liberado por el registro borrado, o que sólo llene una parte.

### 11.6.2 Registros de longitud variable

Los registros de longitud variable surgen de varias maneras en los sistemas de bases de datos:

- Almacenamiento de varios tipos de registros en un mismo archivo.

cabecera				
registro 0	C-102	Navacerrada	400	
registro 1				
registro 2	C-215	Becerril	700	
registro 3	C-101	Centro	500	
registro 4				
registro 5	C-201	Navacerrada	900	
registro 6				
registro 7	C-110	Centro	600	
registro 8	C-218	Navacerrada	700	

**Figura 11.8** El archivo de la Figura 11.5 con la lista libre después del borrado de los registros 1, 4 y 6.

- Tipos de registro que permiten longitudes variables para uno o varios de los campos.
- Tipos de registro que permiten campos repetidos, como los arrays o los multiconjuntos.

Existen diferentes técnicas para implementar los registros de longitud variable.

La **estructura de páginas con ranuras** se utiliza habitualmente para organizar los registros en bloques, y puede verse en la Figura 11.9. Hay una cabecera al principio de cada bloque, que contiene la información siguiente:

1. El número de elementos del registro de la cabecera.
2. El final del espacio vacío del bloque.
3. Un *array* cuyas entradas contienen la ubicación y el tamaño de cada registro.

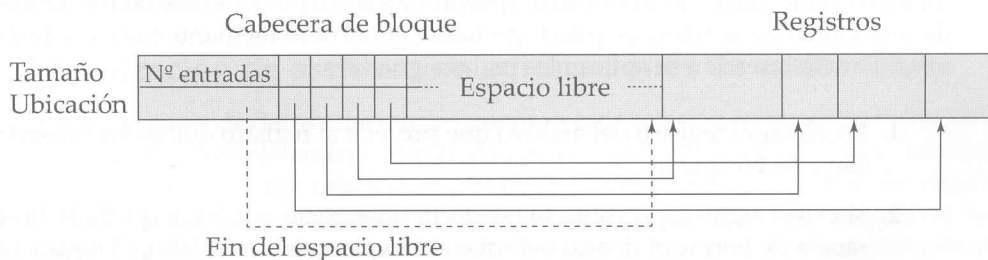
Los registros reales se ubican en el bloque *de manera contigua*, empezando por el final. El espacio libre dentro del bloque es contiguo, entre la última entrada del array de la cabecera y el primer registro. Si se inserta un registro, se le asigna espacio al final del espacio libre y se añade a la cabecera una entrada que contiene su tamaño y su ubicación.

Si se borra un registro, se libera el espacio que ocupa y se da el valor de borrada a su entrada (por ejemplo, se le da a su tamaño el valor de  $-1$ ). Además, se desplazan los registros del bloque situados antes del registro borrado, de modo que se ocupe el espacio libre creado por el borrado y todo el espacio libre vuelve a hallarse entre la última entrada del array de la cabecera y el primer registro. También se actualiza de manera adecuada el puntero de final del espacio libre de la cabecera. Se puede aumentar o disminuir el tamaño de los registros mediante técnicas parecidas, siempre y cuando quede espacio en el bloque. El coste de trasladar los registros no es demasiado elevado, ya que el tamaño del bloque es limitado: un valor habitual es cuatro kilobytes.

La estructura de páginas con ranuras necesita que no haya punteros que señalen directamente a los registros. Por el contrario, los punteros deben apuntar a la entrada de la cabecera que contiene la ubicación verdadera del registro. Este nivel de tratamiento por la dirección permite que se desplacen los registros para evitar la fragmentación del espacio del bloque al tiempo que permite los punteros indirectos al registro.

Las bases de datos almacenan a menudo datos que pueden ser mucho más grandes que los bloques del disco. Por ejemplo, las imágenes o las grabaciones de sonido pueden tener un tamaño de varios megabytes, mientras que los objetos de vídeo pueden llegar a los gigabytes. Recuérdese que SQL soporta los tipos **blob** y **clob**, que almacenan objetos de gran tamaño de los tipos binario y carácter, respectivamente.

La mayor parte de las bases de datos relacionales limitan el tamaño de los registros para que no superen el tamaño de los bloques, con objeto de simplificar la gestión de la memoria intermedia y del espacio libre. Los objetos de gran tamaño y los campos largos suelen guardarse en archivos especiales (o conjuntos de archivos) en lugar de almacenarse con los otros atributos de pequeño tamaño de los registros en los que aparecen. Los objetos de gran tamaño se suelen representar en organizaciones de archivos de árboles  $B^+$ , que se estudian en el Apartado 12.3.4. Estas organizaciones permiten leer el objeto completo o rangos concretos de bytes del mismo, así como insertar y borrar partes del objeto.



**Figura 11.9** Estructura de páginas con ranuras.

## 11.7 Organización de los registros en archivos

Hasta ahora se ha estudiado la manera en que se representan los registros en la estructura de archivo. Las relaciones son conjuntos de registros. Dado un conjunto de registros, la pregunta siguiente es cómo organizarlos en archivos. A continuación se indican varias maneras de organizar los registros en archivos:

- **Organización de los archivos en montículos.** Se puede colocar cualquier registro en cualquier parte del archivo en que haya espacio suficiente. Los registros no se ordenan. Generalmente sólo hay un archivo para cada relación.
- **Organización secuencial de los archivos.** Los registros se guardan en orden secuencial, según el valor de la “clave de búsqueda” de cada uno. El Apartado 11.7.1 describe esta organización.
- **Organización asociativa (hash) de los archivos.** Se calcula una función de asociación (*hash*) para algún atributo de cada registro. El resultado de la función de asociación especifica el bloque del archivo en que se debe colocar cada registro. El Capítulo 12 describe esta organización; está estrechamente relacionada con las estructuras para la creación de índices que se describen en ese capítulo.

Generalmente se emplea un archivo separado para almacenar los registros de cada relación. No obstante, en cada **organización de archivos en agrupaciones de varias tablas** se pueden guardar en el mismo archivo registros de relaciones diferentes; además, los registros relacionados de las diferentes relaciones se guardan en el mismo bloque, por lo que cada operación de E/S afecta a registros relacionados de todas esas relaciones. Por ejemplo, los registros de dos relaciones se pueden considerar relacionados si casan en una reunión de las dos relaciones. El Apartado 11.7.2 describe esta organización.

### 11.7.1 Organización de archivos secuenciales

Los **archivos secuenciales** están diseñados para el procesamiento eficiente de los registros de acuerdo con un orden basado en alguna clave de búsqueda. La **clave de búsqueda** es cualquier atributo o conjunto de atributos; no tiene por qué ser la clave primaria, ni siquiera una superclave. Para permitir la recuperación rápida de los registros según el orden de la clave de búsqueda, éstos se vinculan mediante punteros. El puntero de cada registro señala al siguiente registro según el orden indicado por la clave de búsqueda. Además, para minimizar el número de accesos a los bloques en el procesamiento de los archivos secuenciales, los registros se guardan físicamente en el orden indicado por la clave de búsqueda, o lo más cercano posible.

La Figura 11.10 muestra un archivo secuencial de registros de *cuenta* tomado del ejemplo bancario propuesto. En ese ejemplo, los registros se guardan de acuerdo con el orden de la clave de búsqueda; en este caso, *nombre\_sucursal*.


La organización secuencial de archivos permite que los registros se lean de forma ordenada, lo que puede resultar útil para la visualización, así como para ciertos algoritmos de procesamiento de consultas que se estudian en el Capítulo 13.

Sin embargo, resulta difícil mantener el orden físico secuencial a medida que se insertan y se borran registros, dado que resulta costoso desplazar muchos registros como consecuencia de una sola operación de inserción o de borrado. Se puede gestionar el borrado mediante cadenas de punteros, como ya se ha visto. Para la inserción se aplican las reglas siguientes:

1. Localizar el registro del archivo que precede al registro que se va a insertar según el orden de la clave de búsqueda.
2. Si existe algún registro vacío (es decir, un espacio que haya quedado libre después de una operación de borrado) dentro del mismo bloque que ese registro, el registro nuevo se insertará ahí. En caso contrario, el nuevo registro se insertará en un *bloque de desbordamiento*. En cualquier caso, hay que ajustar los punteros para vincular los registros según el orden de la clave de búsqueda.



C-215	Becerril	750	
C-101	Centro	500	
C-110	Centro	600	
C-305	Collado Mediano	700	
C-217	Galapagar	400	
C-222	Moralzarzal	900	
C-102	Navacerrada	700	
C-201	Navacerrada	700	
C-218	Navacerrada	350	




**Figura 11.10** Archivo secuencial para los registros de *cuenta*.

C-215	Becerril	750	
C-101	Centro	500	
C-110	Centro	600	
C-305	Collado Mediano	700	
C-217	Galapagar	400	
C-222	Moralzarzal	900	
C-102	Navacerrada	700	
C-201	Navacerrada	700	
C-218	Navacerrada	350	

C-888	Leganés	800	
-------	---------	-----	--



**Figura 11.11** El archivo secuencial después de una inserción.

La Figura 11.11 muestra el archivo de la Figura 11.10 después de la inserción del registro (C-888, Leganés, 800). La estructura de la Figura 11.11 permite la inserción rápida de registros nuevos, pero obliga a las aplicaciones de procesamiento de archivos secuenciales a procesar los registros en un orden que no coincide con el físico.

Si hay que guardar un número relativamente pequeño de registros en los bloques de desbordamiento, este enfoque funciona bien. Finalmente, no obstante, la correspondencia entre el orden de la clave de búsqueda y el físico puede perderse totalmente, en cuyo caso el procesamiento secuencial acaba siendo mucho menos eficiente. Llegados a este punto, se debe **reorganizar** el archivo de modo que vuelva a estar físicamente en orden secuencial. Estas reorganizaciones resultan costosas y deben realizarse en momentos en los que la carga del sistema sea baja. La frecuencia con la que las reorganizaciones son necesarias depende de la frecuencia de inserción de registros nuevos. En el caso extremo en que rara vez se produzcan inserciones, es posible mantener siempre el archivo en el orden físico correcto. En ese caso, el campo puntero de la Figura 11.10 no es necesario.

### 11.7.2 Organización de archivos en agrupaciones de varias tablas

Muchos sistemas de bases de datos relacionales guardan cada relación en un archivo diferente, de modo que puedan aprovechar completamente el sistema de archivos proporcionado por el sistema operativo. Generalmente las tuplas de cada relación se pueden representar como registros de longitud fija. Por tanto, se puede hacer que las relaciones se correspondan con una estructura de archivos sencilla. Esta implementación sencilla de los sistemas de bases de datos relacionales resulta adecuada para las implementaciones de bajo coste de las bases de datos como, por ejemplo, los sistemas empotrados o los dispositivos portátiles. En estos sistemas el tamaño de la base de datos es pequeño, por lo que se obtiene

<i>nombre_cliente</i>	<i>número_cuenta</i>
López	C-102
López	C-220
López	C-503
Abril	C-305

Figura 11.12 La relación *impositor*.

poco provecho de una estructura de archivos avanzada. Además, en esos entornos, es fundamental que el tamaño total del código objeto del sistema de bases de datos sea pequeño. Una estructura de archivos sencilla reduce la cantidad de código necesaria para implementar el sistema.

Este enfoque sencillo de la implementación de las bases de datos relacionales resulta menos satisfactorio a medida que aumenta el tamaño de la base de datos. Ya se ha visto que se pueden obtener mejoras en el rendimiento mediante la asignación esmerada de los registros a los bloques y la cuidadosa organización de los propios bloques. Por tanto, resulta evidente que una estructura de archivos más compleja puede resultar beneficiosa, aunque se mantenga la estrategia de guardar cada relación en un archivo diferente.

Sin embargo, muchos sistemas de bases de datos de gran tamaño no utilizan directamente el sistema operativo subyacente para la gestión de los archivos. Por el contrario, se asigna al sistema de bases de datos un archivo de gran tamaño del sistema operativo. En este archivo, el sistema de bases de datos, que también lo administra, guarda todas las relaciones. Para comprender la ventaja de guardar muchas relaciones en un solo archivo considérese la siguiente consulta SQL de la base de datos bancaria:

```
select número_cuenta, nombre_cliente, calle_cliente, ciudad_cliente
from impositor, cliente
where impositor.nombre_cliente = cliente.nombre_cliente
```

Esta consulta calcula una reunión de las relaciones *impositor* y *cliente*. Por tanto, por cada tupla *impositor* el sistema debe encontrar las tuplas de *cliente* con el mismo valor de *nombre\_cliente*. Lo ideal sería poder encontrar estos registros con la ayuda de *índices*, que se estudiarán en el Capítulo 12. Independientemente de la manera en que se encuentren esos registros, hay que transferirlos desde el disco a la memoria principal. En el peor de los casos, cada registro se hallará en un bloque diferente, lo que obligará a efectuar un proceso de lectura de bloque por cada registro necesario para la consulta.

A modo de ejemplo, considérense las relaciones *impositor* y *cliente* de las Figuras 11.12 y 11.13, respectivamente. En la Figura 11.14 se muestra una estructura de archivo diseñada para la ejecución eficiente de las consultas que implican *impositor*  $\bowtie$  *cliente*. Las tuplas *impositor* para cada *nombre\_cliente* se guardan cerca de la tupla *cliente* para el *nombre\_cliente* correspondiente. Esta estructura mezcla las tuplas de dos relaciones, pero permite el procesamiento eficaz de la reunión. Cuando se lee una tupla de la relación *cliente*, se copia del disco a la memoria principal todo el bloque que contiene esa tupla. Dado que las tuplas correspondientes de *impositor* se guardan en el disco cerca de la tupla *cliente*, el bloque que contiene la tupla *cliente* también contiene tuplas de la relación *impositor* necesarias para procesar la consulta. Si un cliente tiene tantas cuentas que los registros de *impositor* no caben en un solo bloque, los registros restantes aparecerán en bloques cercanos.

Una **organización de archivos en agrupaciones de varias tablas** es una organización de archivos, como la mostrada en la Figura 11.14, que almacena registros relacionados de dos o más relaciones en cada bloque. Este tipo de organización de archivos permite leer registros que satisfacen la condición de

<i>nombre_cliente</i>	<i>calle_cliente</i>	<i>ciudad_cliente</i>
López	Mayor	Arganzuela
Abril	Preciados	Valsaín

Figura 11.13 La relación *cliente*.



López	Mayor	Arganzuela
López	C-102	
López	C-220	
López	C-503	
Abril	Preciados	Valsaín
Abril	C-305	

**Figura 11.14** Estructura de archivo en agrupaciones de varias tablas.

reunión en un solo proceso de lectura de bloques. Por tanto, esta consulta concreta se puede procesar de manera más eficiente.

El empleo de la agrupación de varias tablas en un único archivo ha mejorado el procesamiento de una reunión concreta (*impositor*  $\bowtie$  *cliente*) pero retarda el procesamiento de otros tipos de consulta. Por ejemplo:

```
select *
from cliente
```

necesita más accesos a los bloques que con el esquema en el que cada relación se guardaba en un archivo diferente. En lugar de aparecer varios registros de *cliente* en un mismo bloque, cada registro se halla en un bloque diferente. En realidad, lograr hallar todos los registros de *cliente* resulta imposible sin alguna estructura adicional. Para encontrar todas las tuplas de la relación *cliente* en la estructura de la Figura 11.14 se pueden enlazar todos los registros de esa relación mediante punteros, tal y como se muestra en la Figura 11.15.

El empleo de la agrupación de varias tablas depende de los tipos de consulta que el diseñador de la base de datos considere más frecuentes. El uso cuidadoso de la agrupación de varias tablas puede producir mejoras de rendimiento significativas en el procesamiento de las consultas.

## 11.8 Almacenamiento con diccionarios de datos

Hasta ahora sólo se ha considerado la representación de las propias relaciones. Un sistema de bases de datos relacionales necesita tener datos *sobre* las relaciones, como puede ser su esquema. Esta información se denomina **diccionario de datos** o **catálogo del sistema**. Entre los tipos de información que debe guardar el sistema figuran los siguientes:

- El nombre de las relaciones.
- El nombre de los atributos de cada relación.
- El dominio y la longitud de los atributos.
- El nombre de las vistas definidas en la base de datos, y la definición de esas vistas.
- Las restricciones de integridad (por ejemplo, las restricciones de las claves).

López	Mayor	Arganzuela	
López	C-102		
López	C-220		
López	C-503		
Abril	Preciados	Valsaín	
Abril	C-305		

**Figura 11.15** Estructura de archivo con agrupaciones de varias tablas y cadenas de punteros.

*Metadatos\_relación* (nombre\_relación, número\_de\_atributos, organización\_almacenamiento, ubicación)  
*Metadatos\_atributos* (nombre\_atributo, nombre\_relación, tipo\_dominio, posición, longitud)  
*Metadatos\_usuarios* (nombre\_usuario, contraseña\_cifrada, grupo)  
*Metadatos\_índices* (nombre\_índice, nombre\_relación, tipo\_índice, atributos\_índice)  
*Metadatos\_vistas* (nombre\_vista, definición)

**Figura 11.16** Base de datos relacional que representa datos del sistema.

Además, muchos sistemas guardan los siguientes datos de los usuarios del sistema:

- El nombre de los usuarios autorizados.
- La autorización y la información sobre las cuentas de los usuarios.
- Las contraseñas u otra información utilizada para autenticar a los usuarios.

Además, la base de datos puede guardar información estadística y descriptiva sobre las relaciones, como:

- El número de tuplas de cada relación.
- El método de almacenamiento utilizado para cada relación (por ejemplo, con agrupaciones o sin agrupaciones).

El diccionario de datos puede también tener en cuenta la organización del almacenamiento (secuencial, asociativa o en montículos) de las relaciones y la ubicación donde se guarda cada relación:

- Si las relaciones se almacenan en archivos del sistema operativo, el diccionario tendrá en cuenta el nombre del archivo (o archivos) que guarda cada relación.
- Si la base de datos almacena todas las relaciones en un solo archivo, puede que el diccionario tenga en cuenta los bloques que almacenan los registros de cada relación en una estructura de datos como, por ejemplo, una lista enlazada.

En el Capítulo 12, en el que se estudian los índices, se verá que hace falta guardar información sobre cada índice de cada una de las relaciones:

- El nombre del índice.
- El nombre de la relación para la que se crea.
- Los atributos sobre los que se define.
- El tipo de índice formado.

Toda esta información constituye, en efecto, una base de datos en miniatura. Algunos sistemas de bases de datos guardan esta información utilizando código y estructuras de datos especiales. Suele resultar preferible guardar los datos sobre la base de datos en la misma base de datos. Al utilizar la base de datos para guardar los datos del sistema se simplifica la estructura global del sistema y se dedica toda la potencia de la base de datos a obtener un acceso rápido a los datos del sistema.

La elección exacta de la manera de representar los datos del sistema mediante las relaciones debe tomarla el diseñador del sistema. La Figura 11.16 ofrece una representación posible, con las claves primarias subrayadas. En esta representación se da por supuesto que el atributo *atributos\_índice* de la relación *Metadatos\_índices* contiene una lista de uno o varios atributos, que se pueden representar mediante una cadena de caracteres como "*nombre\_sucursal, ciudad\_sucursal*". Por tanto, la relación *Metadatos\_índices* no está en la primera forma normal; se puede normalizar, pero es probable que la representación anterior sea más eficiente en el acceso. El diccionario de datos se suele almacenar de forma no normalizada para conseguir un acceso rápido.