CAPÍTULO

11 ALMACENAMIENTO Y ESTRUCTURA DE ARCHIVOS

En los capítulos anteriores se han destacado los modelos de bases de datos de alto nivel. En el nivel *conceptual* o *lógico* se vieron las bases de datos, en el modelo relacional, como conjuntos de tablas. En realidad, el modelo lógico de las bases de datos es el nivel adecuado en el que se deben centrar los *usuarios*. Esto es por lo que el objetivo de un sistema de bases de datos es simplificar y facilitar el acceso a los datos; los usuarios del sistema no deben someterse sin necesidad alguna a la carga de los detalles físicos del desarrollo del sistema.

En este capítulo, así como en los Capítulos 12, 13 y 14, se desciende desde los niveles superiores y se describen varios métodos para implementar los modelos de datos y los lenguajes presentados en los capítulos anteriores. Se comienza con las características de los medios de almacenamiento subyacentes, como los sistemas de discos y de cintas. Posteriormente se definen varias estructuras de datos que permitirán un acceso rápido a los datos. Se estudian varias arquitecturas alternativas, cada una de ellas más adecuada a diferentes formas de acceder a los datos. La elección final de la estructura de datos hay que hacerla en función del uso que se espera dar al sistema y de las características de cada máquina concreta.

11.1. VISIÓN GENERAL DE LOS MEDIOS FÍSICOS DE ALMACENAMIENTO

En la mayor parte de los sistemas informáticos hay varios tipos de almacenamientos de datos. Estos medios de almacenamiento se clasifican según la velocidad con la que se puede acceder a los datos, por el coste de adquisición del medio por unidad de datos y por la fiabilidad del medio. Entre los medios disponibles habitualmente figuran:

- **Caché.** Caché es la forma de almacenamiento más rápida y costosa. La memoria caché es pequeña; su uso lo gestiona el hardware del sistema informático. No hay que preocuparse sobre la gestión del almacenamiento caché del sistema de bases de datos.
- **Memoria principal.** El medio de almacenamiento utilizado para operar con los datos disponibles es la memoria principal. Las instrucciones de la máquina de propósito general operan en la memoria principal. Aunque la memoria principal puede contener muchos megabites de datos, suele ser demasiado pequeña (o demasiado cara) para guardar toda la base de datos. El contenido de la memoria principal suele perderse si se produce un fallo del suministro eléctrico o una caída del sistema.
- **Memoria *flash*.** También conocida como *memoria sólo de lectura programable y borrable eléctricamente* (*Electrically Erasable Programmable Read-Only Memory*, EEPROM), la memoria *flash* se diferencia de la memoria principal en que los datos pueden sobrevivir a los fallos del suministro eléctrico. La lectura de los datos de la memoria *flash* tarda menos de cien nanosegundos (un nanosegundo es 0,001 milisegundo), lo que resulta aproximadamente igual de rápido que la lectura de los datos de la memoria principal. Sin embargo, la escritura de los datos en la memoria *flash* resulta más complicada (los datos pueden escribirse una sola vez, lo que tarda de cuatro a diez microsegundos, pero no se pueden sobrescribir de manera directa). Para sobreescribir la memoria que se ha escrito previamente hay que borrar simultáneamente todo un banco de memoria; entonces queda preparado para volver a escribir en él. Un inconveniente de la memoria *flash* es que sólo permite un número limitado de ciclos de borrado, que varía entre diez mil y un millón. La memoria *flash* se ha hecho popular como sustituta de los discos magnéticos para guardar pequeños volúmenes de datos (de cinco a diez megabytes) en los sistemas informáticos de coste reducido que se incluyen en otros dispositivos, como computadoras de bolsillo y en otros dispositivos electrónicos como cámaras digitales.
- **Almacenamiento en discos magnéticos.** El principal medio de almacenamiento a largo plazo de datos en conexión es el disco magnético. Generalmente se guarda en este tipo de discos toda la base de datos. Para tener acceso a los datos hay que tras-

ladarlos desde el disco a la memoria principal. Después de realizar la operación hay que escribir en el disco los datos que se han modificado.

El tamaño de los discos magnéticos actuales oscila entre unos pocos gigabytes y 80 gigabytes. Tanto el extremo superior como el inferior de este rango han ido creciendo a un ritmo del 50 por ciento anual, y se pueden esperar discos de mucha mayor capacidad cada año. El almacenamiento en disco puede aguantar los fallos del suministro eléctrico y caídas del sistema. Los dispositivos de almacenamiento en disco pueden fallar a veces y, en consecuencia, destruir los datos, pero tales fallos suelen producirse con mucha menos frecuencia que una caída del sistema.

- **Almacenamiento óptico.** La forma más popular de almacenamiento óptico es el disco compacto (*Compact Disk, CD*), que puede almacenar alrededor de 640 megabytes de datos, y el *disco de video digital* (*Digital Video Disk, DVD*), que puede almacenar 4,7 u 8,5 gigabytes de datos por cada cara del disco (o hasta 17 gigabytes en un disco de doble cara). Los datos se almacenan en un disco por medios ópticos y se leen mediante un láser. Los discos ópticos usados en los discos compactos de sólo lectura (CD-ROM) o en los discos de vídeo digital de sólo lectura (DVD-ROM) no se pueden escribir, pero se suministran con datos pregrabados.

  Hay versiones «grabar una vez» de los discos compactos (llamada CD-R) y de los discos de vídeo digital (llamada DVD-R), que se pueden escribir sólo una vez; estos disco también se denominan **de escritura única y lectura múltiple** (*Write-Once, Read-Only Memory*, WORM). También hay versiones «escribir varias veces» de los discos compactos (llamada CD-RW) y de los discos de vídeo digital (DVD-RW y DVD-RAM), que permiten escribir varias veces. Los discos compactos grabables son dispositivos de almacenamiento magnetoópticos que usan medios ópticos para leer datos codificados magnéticamente. Estos discos son útiles para el almacenamiento de archivos así como para la distribución de datos.

  Los *cambiadores automáticos* (*jukebox*) contienen unas cuantas unidades y numerosos discos que pueden cargarse de manera automática en una de las unidades (mediante un brazo robotizado) a petición del usuario.

- **Almacenamiento en cinta.** El almacenamiento en cinta se utiliza principalmente para copias de seguridad y datos de archivo. Aunque la cinta magnética es mucho más barata que los discos, el acceso a los datos resulta mucho más lento, ya que el acceso a la cinta debe ser secuencial desde su comienzo. Por este motivo, el almacenamiento se denomina almacenamiento de **acceso secuencial**. En cambio, el almacenamiento en disco se denomina almacenamiento de acceso directo porque es posible leer datos desde cualquier ubicación del disco.

  Las cintas tienen una capacidad elevada (actualmente hay disponibles cintas de 40 a 300 gigabytes) y pueden retirarse de la unidad de lectura, lo que facilita un almacenamiento de coste reducido para archivos. Los cambiadores automáticos de cinta se utilizan para guardar conjuntos de datos excepcionalmente grandes, como los datos de sensores remotos de los satélites, que pueden alcanzar cientos de terabytes (10^{12} bytes) o incluso petabytes (10^{15} bytes) de datos.

Los diferentes medios de almacenamiento pueden organizarse en una jerarquía (Figura 11.1) de acuerdo con su velocidad y su coste. Los niveles superiores son de coste elevado, pero rápidos. A medida que se desciende por la jerarquía el coste por bit disminuye, mientras que el tiempo de acceso aumenta. Este compromiso es razonable; si un sistema de almacenamiento dado fuera a la vez más rápido y menos costoso que otro (siendo iguales las demás propiedades) no habría ninguna razón para utilizar la memoria más lenta y más costosa. De hecho, muchos dispositivos de almacenamiento primitivos, incluyendo la cinta de papel y las memorias de núcleos de ferrita, se hallan relegados a los museos ahora que la cinta magnética y la memoria de semiconductores se han vuelto más rápidas y económicas. Las propias cintas magnéticas se utilizaron para guardar los datos activos cuando los discos resultaban costosos y tenían una capacidad de almacenamiento reducida. Hoy en día casi todos los datos activos se almacenan en discos, excepto en los raros casos en que se guardan en cambiadores automáticos de cinta u ópticos.

Los medios de almacenamiento más rápidos (por ejemplo, caché y memoria principal) se denominan **almacenamiento primario**. Los medios del siguiente

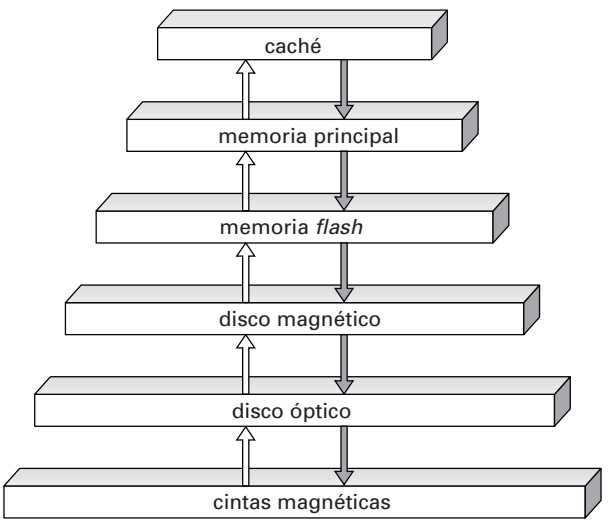


FIGURA 11.1. Jerarquía de dispositivos de almacenamiento.

nivel de la jerarquía (por ejemplo, los discos magnéticos) se conocen como **almacenamiento secundario** o **almacenamiento en conexión**. Los medios del nivel inferior de la jerarquía —por ejemplo, cinta magnética y los cambiadores automáticos de discos ópticos— se denominan **almacenamiento terciario** o **almacenamiento sin conexión**.

Además de la velocidad y del coste de los diferentes sistemas de almacenamiento está también el asunto de la volatilidad del almacenamiento. El **almacenamiento volátil** pierde su contenido cuando se suprime el suministro eléctrico del dispositivo. En la jerarquía mostrada en la Figura 11.1 los sistemas de almacenamiento desde la memoria principal hacia arriba son volátiles, mientras que los sistemas de almacenamiento por debajo de la memoria principal son no volátiles. En ausencia de los costosos sistemas de baterías y de sistemas de generadores de reserva, los datos deben escribirse en **almacenamiento no volátil** por razones de seguridad. Se volverá a este asunto en el Capítulo 17.

11.2. DISCOS MAGNÉTICOS

Los discos magnéticos constituyen el principal medio de almacenamiento secundario en los sistemas informáticos modernos. Las capacidades de los discos han estado creciendo alrededor del 50 por ciento anual, pero los requisitos de las grandes aplicaciones también han estado creciendo muy rápido, en algunos casos más que la tasa de crecimiento de las capacidades de los discos. Una base de datos comercial grande típica puede necesitar centenares de discos.

11.2.1. Características físicas de los discos

Físicamente, los discos son relativamente sencillos (Figura 11.2). Cada **plato** del disco tiene una forma circular plana. Sus dos superficies están cubiertas por un material magnético y la información se graba en ellas. Los platos están hechos de metal rígido o de vidrio y están cubiertos (generalmente por los dos lados) con material magnético para grabaciones. Estos discos magnéticos se denominan **discos rígidos** para distinguirlos de los **disquetes**, que están hechos de material flexible.

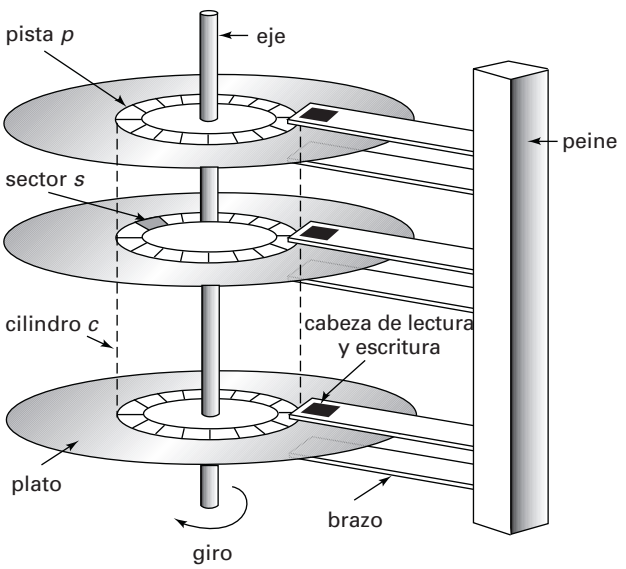


FIGURA 11.2. Mecanismo de disco de cabezas móviles.

Mientras se está utilizando el disco, un motor lo hace girar a una velocidad constante elevada (generalmente 60, 90 o 120 revoluciones por segundo, pero también hay disponibles discos girando a 250 revoluciones por segundo). Hay una cabeza de lectura y escritura ubicada justo encima de la superficie del plato. La superficie del disco se divide a efectos lógicos en **pistas**, que se subdividen en **sectores**. Un **sector** es la unidad mínima de información que puede leerse o escribirse en el disco. En los discos disponibles actualmente, los tamaños de sector son normalmente de 512 bytes; hay alrededor de 16.000 pistas en cada disco y de dos a cuatro platos por disco. Las pistas internas (más cercanas al eje) son de menor longitud, y en los disco de la generación actual, las pistas exteriores contienen más sectores que las pistas internas; normalmente hay alrededor de 200 sectores por pista en las pistas internas y alrededor de 400 sectores por pistas en las pistas externas. Estos números pueden variar entre diferentes modelos; los modelos de mayor capacidad tienen más sectores por pista y más pistas en cada plato.

La **cabeza de lectura y escritura** guarda magnéticamente la información en los sectores en forma de inversiones de la dirección de magnetización del material magnético.

Cada cara de un plato del disco tiene una cabeza de lectura y escritura que se desplaza por el plato para tener acceso a las diferentes pistas. Un disco suele contener muchos platos y las cabezas de lectura y escritura de todas las pistas están montadas en un solo dispositivo denominado **brazo del disco** y se desplazan conjuntamente. El conjunto de los platos del disco montados sobre un eje y las cabezas montadas en el brazo del disco se denomina **dispositivo cabeza-disco**. Dado que las cabezas de todos los platos se desplazan conjuntamente, cuando la cabeza de un plato se halle en la pista *i*-ésima, las cabezas de todos las demás platos también se encontrarán en la pista *i*-ésima de sus platos respectivos. Por consiguiente, las pistas *i*-ésimas de todos los platos se denominan conjuntamente **cilindro** *i*-ésimo.

Actualmente, los discos con un diámetro de plato de 3 $^1/_2$ pulgadas dominan el mercado. Tienen un menor coste y tiempos de búsqueda más cortos (debido a las menores distancias de búsqueda) que los discos de tamaño mayor (de hasta 14 pulgadas) que fueron habituales en el pasado, y proporcionan una alta capacidad de almacenamiento.

Las cabezas de lectura y escritura se mantienen tan próximas como sea posible a la superficie de los discos para aumentar la densidad de grabación. A menudo la cabeza flota o vuela a sólo micras de la superficie del disco; las revoluciones del disco crean una pequeña brisa y el dispositivo de cabezas se manufactura de forma que la brisa mantenga la cabeza flotando sobre la superficie del disco. Como la cabeza flota tan cercana a la superficie, los platos deben estar elaborados esmeradamente para que sean lisos. Los choques de las cabezas pueden suponer un problema. Si la cabeza contacta con la superficie del disco, arrancará del disco el medio de grabación, destruyendo los datos que hubiera allí. Generalmente, la cabeza que toca la superficie hace que el material arrancado flote en el aire y se coloque entre las cabezas y sus platos, lo que causa más choques. En circunstancias normales un choque de las cabezas da lugar a que falle todo el disco y haya que sustituirlo. Los dispositivos de disco de la generación actual usan una fina capa de metal magnético como medio de grabación. Son menos susceptibles de fallar a causa de choques de las cabezas que los antiguos discos cubiertos de óxido.

Un *disco de cabezas fijas* tiene una cabeza diferente para cada pista. Esta disposición permite a la computadora cambiar rápidamente de pista a pista sin tener que desplazar el dispositivo de las cabezas, pero necesita gran número de cabezas, lo que hace al dispositivo excesivamente costoso. Algunos sistemas de discos tienen varios brazos de disco, lo que permite que se tenga acceso simultáneamente a más de una pista del mismo plato. Los discos de cabezas fijas y los discos con varios brazos se utilizaban en grandes sistemas de alto rendimiento, pero ya no se fabrican.

Un **controlador de disco** actúa como interfaz entre el sistema informático y el hardware concreto de la unidad de disco. Acepta las órdenes de alto nivel para leer o escribir en un sector e inicia las acciones, como desplazar el brazo del disco a la pista adecuada y leer o escribir realmente los datos. Los controladores de disco también añaden **comprobación de suma** a cada sector en el que se escribe; la comprobación de suma se calcula a partir de los datos escritos en el sector. Cuando se vuelve a leer el sector, se vuelve a calcular la suma a partir de los datos recuperados y se compara con la suma de comprobación guardada; si los datos se han deteriorado resulta muy probable que la suma de comprobación recién calculada no coincida con la guardada. Si se produce un error de este tipo, el controlador volverá a intentar varias veces la lectura; si el error sigue produciéndose, el controlador indicará un fallo de lectura.

Otra labor interesante llevada a cabo por los controladores de disco es la **reasignación de los sectores dañados**. Si el controlador detecta que un sector está dañado cuando se da formato al disco por primera vez, o cuando se realiza un intento de escribir en el sector, puede reasignar lógicamente el sector a una ubicación física diferente (escogida de entre un grupo de sectores extra preparado con esta finalidad). La reasignación se anota en disco o en memoria no volátil y la escritura se realiza en la nueva ubicación.

En la Figura 11.3 se muestra la manera en que los discos se conectan a un sistema informático. Al igual que otras unidades de almacenamiento, los discos se conectan a un sistema informático o a un controlador mediante una conexión de alta velocidad. En los sistemas de disco modernos, las funciones de menor nivel del controlador de disco, como el control del brazo, el cálculo y verificación de la comprobación de suma y la reasignación de los sectores dañados se implementan en la unidad de disco.

La interfaz **ATA (AT Attachment)** (que es una versión más rápida que la **interfaz electrónica de dispositivos integrados** [IDE, Integrated Drive Electronics] usada antiguamente en los PC de IBM) y la **interfaz de conexión para sistemas informáticos pequeños** (*SCSI, Small Computer-System Interconnect Interface*, pronunciado «escasi») se usan habitualmente para conectar los discos con las computadoras personales y las estaciones de trabajo. Los grandes sistemas y los siste-

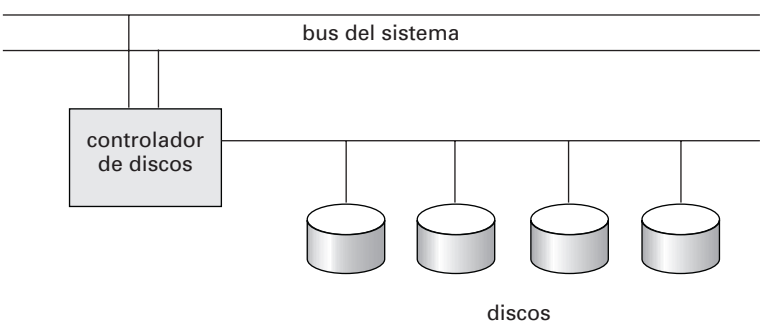


FIGURA 11.3. Subsistema de disco.

mas servidores suelen disponer de una interfaz más rápida y cara, como las versiones de alta capacidad de la interfaz SCSI, y la interfaz Fibre Channel.

Los discos se conectan por lo general directamente al controlador de disco mediante cables, pero también pueden estar situados en una ubicación remota y estar conectados mediante una red de alta velocidad al controlador de disco. En una arquitectura de **red de área de almacenamiento** (Storage-Area Network, SAN), se conecta un gran número de discos mediante una red de alta velocidad a varias computadoras servidoras. Los discos generalmente se organizan localmente usando una técnica de organización del almacenamiento denominada «disposición redundante de discos independientes (Redundant Array of Independent Disks, RAID)» (descritos posteriormente en el Apartado 11.3) para dar a los servidores una vista lógica de un disco de gran tamaño y muy fiable. El controlador y el disco usan las interfaces SCSI y Fibre Channel para comunicarse entre sí, aunque puedan estar separados por una red. El acceso remoto a los discos también significa que los discos que contienen datos importantes se pueden guardar en una habitación del servidor central donde se pueden supervisar y mantener por los administradores del sistema, en lugar de estar esparcidos en diferentes lugares de la organización.

11.2.2. Medidas del rendimiento de los discos

Las principales medidas de la calidad de un disco son la capacidad, el tiempo de acceso, la velocidad de transferencia de datos y la fiabilidad.

El **tiempo de acceso** es el tiempo transcurrido desde que se formula una solicitud de lectura o de escritura hasta que comienza la transferencia de datos. Para tener acceso (es decir, para leer o escribir) a los datos en un sector dado del disco, primero se debe desplazar el brazo para que se ubique sobre la pista correcta y luego hay que esperar a que el sector aparezca bajo el brazo por acción de la rotación del disco. El tiempo para volver a ubicar el brazo se denomina **tiempo de búsqueda** y aumenta con la distancia que deba recorrer el brazo. Los tiempos de búsqueda típicos varían de dos a treinta milisegundos, en función de la distancia de la pista a la posición inicial del brazo. Los discos de menor tamaño tienden a tener tiempos de búsqueda menores, dado que la cabeza tiene que recorrer una distancia menor.

El **tiempo medio de búsqueda** es la media de los tiempos de búsqueda medido en una sucesión de solicitudes aleatorias (uniformemente distribuidas). Si todas las pistas tienen el mismo número de sectores y despreciando el tiempo requerido para que la cabeza inicie su movimiento y lo detenga, se puede demostrar que el tiempo medio de búsqueda es un tercio del peor de los tiempos de búsqueda posibles. Teniendo en cuenta estos factores, el tiempo medio de búsqueda es alrededor de la mitad del tiempo máximo de búsqueda. Los tiempos medios de búsqueda varían actualmente entre cuatro y diez milisegundos, dependiendo del modelo de disco.

Una vez ha tenido lugar la búsqueda, el tiempo que se pasa esperando a que el sector al que hay que tener acceso aparezca bajo la cabeza se denomina **tiempo de latencia rotacional**. Las velocidades rotacionales típicas de los discos actuales varían de 5.400 rotaciones por minuto (90 rotaciones por segundo) hasta 15.000 rotaciones por minuto (250 rotaciones por segundo) o, lo que es lo mismo, de 4 a 11,1 milisegundos por rotación. En media hace falta la mitad de una rotación del disco para que aparezca bajo la cabeza el comienzo del sector deseado. Por tanto, el **tiempo de latencia medio** del disco es la mitad del tiempo empleado en una rotación completa del disco.

El tiempo de acceso es la suma del tiempo de búsqueda y del tiempo de latencia y varía de 8 a 20 milisegundos. Una vez que se ha ubicado bajo la cabeza el primer sector de datos, comienza su transferencia. La **velocidad de transferencia de datos** es la velocidad a la que se pueden recuperar o guardar datos en el disco. Los sistemas de disco actuales anuncian que permiten velocidades máximas de transferencia de 25 a 40 megabytes por segundo, aunque las velocidades de transferencia reales pueden ser significativamente menores, alrededor de 4 a 8 megabytes por segundo.

La última de las medidas de los discos utilizadas con frecuencia es el **tiempo medio entre fallos**, que es una medida de la fiabilidad del disco. El tiempo medio entre fallos de un disco (o de cualquier otro sistema) es la cantidad de tiempo que, en media, se puede esperar que el sistema funcione de manera continua sin tener ningún fallo. De acuerdo con los anuncios de los fabricantes, el tiempo medio entre fallos de los discos actuales varía de 30.000 a 1.200.000 horas (de 3,4 a 136 años). En la práctica, el tiempo medio entre fallos anunciado se calcula en términos de la probabilidad de fallo cuando el disco es nuevo (este escenario significa que dados 1.000 discos relativamente nuevos, si el tiempo medio entre fallos es 1.200.000 horas, uno de ellos fallará en media en 1.200 horas). Un tiempo medio entre fallos de 1.200.000 horas no implica que se espere que el disco vaya a funcionar 136 años. La mayoría de los discos tienen un periodo de vida esperado de cinco años, y tienen significativamente más fallos cuando son algunos años más viejos.

Puede haber varios discos compartiendo una interfaz de discos. La norma de la interfaz ATA-4 ampliamente usada (también conocida como Ultra-DMA) soporta velocidades de transferencia de 33 megabytes por segundo, mientras que ATA-5 soporta 66 megabytes por segundo. SCSI-3 (Ultra 2 wide SCSI) soporta 40 megabytes por segundo, mientras que la interfaz más cara Fibre Channel soporta hasta 256 megabytes por segundo. La velocidad de transferencia de la interfaz se comparte entre todos los discos conectados a la interfaz.

11.2.3. Optimización del acceso a los bloques del disco

Las solicitudes de E/S al disco las generan tanto el sistema de archivos como el gestor de la memoria virtual que se halla en la mayor parte de los sistemas operativos. Cada solicitud especifica la dirección del disco a la que hay que hacer referencia; esa dirección está en la forma de un *número de bloque*. Un **bloque** es una secuencia continua de sectores de una sola pista de un plato. Los tamaños de los bloques varían de 512 bytes a varios kilobytes. Los datos se transfieren entre el disco y la memoria principal en unidades de bloques. Los niveles inferiores del gestor del sistema de archivos transforman las direcciones de los bloques en cilindro, superficie y número de sector del nivel de hardware.

Dado que el acceso a los datos del disco es varios órdenes de magnitud más lento que el acceso a la memoria principal, se ha prestado mucha atención a la mejora de la velocidad de acceso a los bloques del disco. La utilización de memoria intermedia para bloques en la memoria para satisfacer las solicitudes futuras se discute en el Apartado 11.5. Aquí se discuten otras técnicas.

- **Planificación.** Si hay que transferir varios bloques de un cilindro desde el disco a la memoria principal puede que se logre disminuir el tiempo de acceso solicitando los bloques en el orden en el que pasarán por debajo de las cabezas. Si los bloques deseados se hallan en cilindros diferentes resulta ventajoso solicitar los bloques en un orden que minimice el movimiento del brazo del disco. Los algoritmos de **planificación del brazo del disco** intentan ordenar el acceso a las pistas de manera que se aumente el número de accesos que puede procesarse. Un algoritmo utilizado con frecuencia es el **algoritmo del ascensor**, que funciona de manera parecida a muchos ascensores. Supóngase que, inicialmente, el brazo se desplaza desde la pista más interna hacia el exterior del disco. Bajo el control del algoritmo del ascensor, en cada pista para la que hay una solicitud de acceso el brazo se detiene, atiende las solicitudes para la pista y continúa desplazándose hacia el exterior hasta que no queden solicitudes pendientes para las pistas más externas. En este punto el brazo cambia de dirección, se desplaza hacia el interior y vuelve a detenerse en cada pista para las que haya solicitudes hasta alcanzar una pista en la que no haya solicitudes para pistas más cercanas al centro del disco. En ese momento cambia de dirección e inicia un nuevo ciclo. Los controladores de disco suelen realizar la labor de reordenar las solicitudes de lectura para mejorar el rendimiento, dado que conocen perfectamente la organización de los bloques del disco, la posición rotacional de los platos y la posición del brazo.
- **Organización de archivos.** Para reducir el tiempo de acceso a los bloques se pueden organizar los bloques del disco de una manera que se corresponda fielmente con la forma en que se espera tener acceso a los datos. Por ejemplo, si se espera tener acceso secuencial a un archivo, en teoría se deberían guardar secuencialmente en cilindros adyacentes todos los bloques del archivo. Los sistemas operativos más antiguos, como los sistemas operativos de IBM para grandes sistemas proporcionaban a los programadores un control detallado de la ubicación de los archivos, lo que permitía a los programadores reservar un conjunto de cilindros para guardar un archivo. Sin embargo, este control supone una carga para el programador o el administrador del sistema que debe decidir, por ejemplo, los cilindros que ha de asignar a un archivo y puede exigir una reorganización costosa si se insertan o borran datos del archivo.

  Los sistemas operativos posteriores, como Unix y los sistemas operativos para computadoras personales ocultan a los usuarios la organización del disco y gestionan la asignación de manera interna. Sin embargo, en el transcurso del tiempo, un archivo secuencial puede quedar **fragmentado**; es decir, sus bloques pueden quedar dispersos por el disco. Para reducir la fragmentación, el sistema puede hacer una copia de seguridad de los datos del disco y restaurar todo el disco. La operación de restauración vuelve a escribir de manera contigua (o casi) los bloques de cada archivo. Algunos sistemas (como MS-DOS) disponen de utilidades que examinan el disco y luego desplazan los bloques para reducir la fragmentación. Los aumentos de rendimiento obtenidos con estas técnicas pueden ser elevados, pero no se suele poder utilizar el sistema mientras se están ejecutando estas utilidades.

- **Memoria intermedia de escritura no volátil.** Dado que el contenido de la memoria se pierde durante los fallos de suministro eléctrico, hay que guardar en disco la información sobre las actualizaciones de las bases de datos para que superen las posibles caídas del sistema. Por tanto, el rendimiento de las aplicaciones de bases de datos sensibles a las actualizaciones, como los sistemas de procesamiento de transacciones, dependen mucho de la velocidad de escritura en el disco.

  Se puede utilizar **memoria no volátil de acceso aleatorio** (RAM no volátil) para acelerar la escritura en el disco de manera drástica. El contenido de la RAM no volátil no se pierde durante un fallo del suministro eléctrico. Una manera habitual de implementar la RAM no volátil es utilizar RAM alimentada por baterías. La idea es que, cuando el sistema de bases de datos (o el sistema operativo) solicita que se escriba un bloque en el disco, el controlador del disco escriba el bloque en una memoria intermedia de RAM no volátil y comunique de manera inmediata al sistema ope-

rativo que la escritura se completó con éxito. El controlador escribe los datos en su destino en el disco en cualquier momento en que el disco no tenga otras solicitudes o cuando la memoria intermedia de RAM no volátil se llene. Cuando el sistema de bases de datos solicita la escritura de un bloque sólo percibe un retraso si la memoria intermedia de RAM no volátil se encuentra llena. Durante la recuperación de una caída del sistema se vuelven a escribir en el disco todas las escrituras que se hallan pendientes en la memoria intermedia de RAM no volátil.

El grado en el que la RAM no volátil aumenta el rendimiento se ilustra en el siguiente ejemplo. Supóngase que las solicitudes de escritura se reciben de manera aleatoria y que el disco se halla ocupado el 90 por ciento del tiempo como media[1]. Si se dispone de una memoria intermedia de RAM no volátil de cincuenta bloques, las solicitudes de escritura sólo encontrarán la memoria intermedia llena, en media, una vez por minuto (y tendrán, por tanto, que esperar a que concluya un proceso de escritura en el disco). Duplicar la memoria intermedia a cien bloques resulta en encontrar la memoria intermedia llena sólo una vez por hora. Por tanto, en la mayoría de los casos, las escrituras a disco se pueden ejecutar sin que el sistema de bases de datos espere debido a una búsqueda o a la latencia rotacional.

- **Disco de registro histórico.** Otro enfoque para reducir las latencias de escritura es utilizar un *disco de registro histórico* (de manera muy parecida a la memoria intermedia RAM no volátil). Todos los accesos al disco de registro histórico son secuenciales, lo que elimina principalmente el tiempo de búsqueda y, así, pueden escribirse simultáneamente varios bloques consecutivos, lo que hace que los procesos de escritura en el disco sean varias veces más rápidos que los procesos de escritura aleatorios. Igual que ocurría anteriormente también hay que escribir los datos en su ubicación verdadera en el disco, pero este proceso de escritura puede llevarse a cabo sin que el sistema de bases de datos tenga que esperar a que se complete. Más aún, el disco de registro histórico puede reordenar las escrituras para reducir el movimiento del brazo. Si el sistema cae antes de que se hayan realizado algunas escrituras en la ubicación real del disco, cuando el sistema se recupere lee el disco de registro histórico para encontrar las escrituras que no se han realizado y entonces las completa.

  Los sistemas de archivo que soportan los discos de registro histórico mencionados se denominan **sistemas de archivos de diario**. Los sistemas de archivos de diario se pueden implementar incluso sin un disco de registro histórico aislado, guardando los datos y el registro histórico en el mismo disco. Al hacerlo así se reduce el coste económico a expensas de un menor rendimiento.

  El **sistema de archivos basado en registro histórico** es una versión extrema del enfoque del disco de registro histórico. Los datos no se vuelven a escribir en su ubicación original en el disco; en su lugar, el sistema de archivos hace un seguimiento del lugar del disco de registro histórico en que se escribieron los bloques más recientemente y los recupera desde esa ubicación. El propio disco de registro histórico se compacta de manera periódica, por lo que se pueden eliminar los procesos de escritura antiguos que se han sobrescrito posteriormente. Este enfoque mejora el rendimiento de los procesos de escritura pero genera un elevado grado de fragmentación de los archivos que se actualizan con frecuencia. Como se señaló anteriormente, esa fragmentación aumenta el tiempo de búsqueda en la lectura secuencial de los archivos.

11.3. RAID

Los requisitos de almacenamiento de datos de algunas aplicaciones (en particular las aplicaciones Web, de bases de datos y multimedia) han estado creciendo tan rápidamente que se necesita un gran número de discos para almacenar sus datos, incluso aunque las capacidades de los discos hayan estado creciendo muy rápidamente.

Tener un gran número de discos en un sistema presenta oportunidades para mejorar la velocidad a la que se pueden leer o escribir los datos si los discos funcionan en paralelo. El paralelismo se puede usar para realizar varias lecturas o escrituras independientes simultáneamente. Además, esta configuración ofrece la posibilidad de mejorar la fiabilidad del almacenamiento de datos, ya que se puede guardar información repetida en varios discos. Por tanto, el fallo de un disco no provoca una pérdida de datos.

Para abordar los problemas de rendimiento y de fiabilidad se han propuesto varias técnicas de organización de discos, denominadas colectivamente **disposición redundante de discos independientes** (**RAIDs - Redundant Array of Independent Disks**), para conseguir un rendimiento y fiabilidad mejorados.

[1] Para el lector interesado en la estadística se asume una distribución de Poisson para las llegadas. Las tasas exactas de llegada y de servicio no se necesitan, dado que el uso del disco proporciona suficiente información para estos cálculos.

En el pasado, los diseñadores de sistemas vieron los sistemas de almacenamiento compuestos de discos pequeños y de bajo coste como una alternativa económicamente efectiva a los discos grandes y caros; el coste por megabyte de los discos más pequeños era menor que el de los discos mayores. De hecho, la I de RAID, que ahora representa *independientes*, originalmente representaba *económicos* (*inexpensive*). Hoy en día, sin embargo, todos los discos son pequeños físicamente, y los discos de gran capacidad tienen realmente un menor coste por megabyte. Los sistemas RAID se usan por su mayor fiabilidad y por su mayor velocidad de transferencia de datos, más que por motivos económicos.

11.3.1. Mejora de la fiabilidad mediante la redundancia

Considérese en primer lugar la fiabilidad. La posibilidad de que algún disco de una disposición de *N* discos falle es mucho más elevada que la posibilidad de que un único disco concreto falle. Supóngase que el tiempo medio entre fallos de un solo disco es de cien mil horas o, aproximadamente, once años. Por tanto, el tiempo medio entre fallos de un disco de una disposición de cien discos será de 100.000/100 = 1.000 horas, o 42 días, ¡lo que no es mucho! Si sólo se guarda una copia de los datos, cada fallo de un disco dará lugar a la pérdida de una cantidad de datos significativa (como se discutió anteriormente en el Apartado 11.2.1). Una tasa de pérdida de datos tan elevada resulta inaceptable.

La solución al problema de la fiabilidad es introducir la **redundancia**; es decir, se guarda información adicional que normalmente no se necesita pero que puede utilizarse en caso de fallo de un disco para reconstruir la información perdida. Por tanto, incluso si falla un disco, los datos no se pierden, por lo que el tiempo medio efectivo entre fallos aumenta, siempre que se cuenten sólo los fallos que den lugar a pérdida de datos o a su no disponibilidad.

El enfoque más sencillo (pero el más costoso) para la introducción de la redundancia es duplicar todos los discos. Esta técnica se denomina **creación de imágenes** (o, a veces, *creación de sombras*). Un disco lógico consiste, por tanto, en dos discos físicos y cada proceso de escritura se lleva a cabo en ambos discos. Si uno de los discos falla se pueden leer los datos del otro. Los datos sólo se perderán si falla el segundo disco antes de que se repare el primero que falló.

El tiempo medio entre fallos (donde fallo es la pérdida de datos) de un disco con imagen depende del tiempo medio entre fallos de cada disco y del **tiempo medio de reparación**, que es el tiempo que se tarda (en media) en sustituir un disco averiado y en restaurar sus datos. Supóngase que los fallos de los dos discos son *independientes*; es decir, no hay conexión entre el fallo de un disco y el del otro. Por tanto, si el tiempo medio entre fallos de un solo disco es de cien mil horas y el tiempo medio de reparación es de diez horas el **tiempo medio entre pérdidas de datos** de un sistema de discos con imagen es $100.000^2/(2 \times 10) = 500 \times 10^6$ horas, o ¡57.000 años! (aquí no se entra en detalle en los cálculos; se proporcionan en las referencias de las notas bibliográficas).

Hay que tener en cuenta que la suposición de la independencia de los fallos de los discos no resulta válida. Los fallos en el suministro eléctrico y los desastres naturales como los terremotos, los incendios y las inundaciones pueden dar lugar a daños simultáneos de ambos discos. A medida que los discos envejecen, la probabilidad de fallo aumenta, lo que aumenta la posibilidad de que falle el segundo disco mientras se repara el primero. A pesar de todas estas consideraciones, sin embargo, los sistemas de discos con imagen ofrecen una fiabilidad mucho más elevada que los sistemas de discos únicos. Hoy en día están disponibles sistemas de discos con imagen con un tiempo medio entre pérdidas de datos de entre quinientas mil y un millón de horas, o de cincuenta y cinco a ciento diez años.

Los fallos en el suministro eléctrico son una causa especial de preocupación, dado que tienen lugar mucho más frecuentemente que los desastres naturales. Los fallos en el suministro eléctrico no son un problema si no se está realizando ninguna transferencia de datos al disco cuando tienen lugar. Sin embargo, incluso con la creación de imágenes de los discos, si los procesos de escritura se hallan en curso en el mismo bloque en ambos discos y el suministro eléctrico falla antes de que ambos bloques estén escritos completamente, los dos bloques pueden quedar en un estado inconsistente. La solución a este problema es escribir una copia en primer lugar y luego la otra, de modo que siempre sea consistente una de las copias. Hacen falta algunas acciones adicionales cuando se vuelve a iniciar el sistema después de un fallo en el suministro eléctrico para recuperar los procesos de escritura incompletos. Este asunto se examina en el Ejercicio 11.4.

11.3.2. Mejora del rendimiento mediante el paralelismo

Considérense ahora las ventajas del acceso en paralelo a varios discos. Con la creación de imágenes de los discos la velocidad a la que las solicitudes de lectura pueden procesarse se duplica, dado que las solicitudes de lectura pueden enviarse a cualquiera de los discos (siempre y cuando los dos discos de la pareja estén operativos, como es el caso habitual). La velocidad de transferencia de cada proceso de lectura es la misma que en los sistemas de discos únicos, pero el número de procesos de lectura por unidad de tiempo se ha duplicado.

Con varios discos también se puede mejorar la velocidad de transferencia **distribuyendo los datos** entre varios discos. En su forma más sencilla la distribución de datos consiste en dividir los bits de cada byte entre varios discos; esta distribución se denomina **distribución en el nivel de bit**. Por ejemplo, si se dispone de una disposición de ocho discos se puede escribir el bit *i* de cada byte en el disco *i*. La disposición de ocho dis-

cos puede tratarse como un solo disco con sectores que tienen ocho veces el tamaño normal y, lo que es más importante, que tienen ocho veces la velocidad de acceso. En una organización así cada disco toma parte en todos los accesos (de lectura o de escritura) por lo que el número de accesos que pueden procesarse por segundo es aproximadamente el mismo que en un solo disco, pero cada acceso puede leer ocho veces tantos datos por unidad de tiempo como con un solo disco. La distribución en el nivel de bit puede generalizarse a cualquier número de discos que sea múltiplo o divisor de ocho. Por ejemplo, si se utiliza una disposición de cuatro discos, los bits i y $4 + i$ de cada byte irán al disco i.

La *distribución en el nivel de bloque* reparte los bloques entre varios discos. Trata la disposición de discos como un único y gran disco, y proporciona números lógicos a los bloques; se asume que los números de bloque comienzan en 0. Con una disposición de n discos, la distribución en el nivel de bloque asigna el bloque lógico i de la disposición de discos al disco (i mod n) + 1; usa el bloque físico $\lfloor i/n \rfloor$ – ésimo del disco para almacenar el bloque lógico i. Por ejemplo, con ocho discos, el bloque lógico 0 se almacena el bloque físico 0 del disco 1, mientras que el bloque lógico 11 se almacena en el bloque físico 1 del disco 4. Al leer un archivo grande, la distribución en el nivel de bloque busca n bloques en un instante en paralelo en los n discos, dando una gran velocidad de transferencia para grandes lecturas. Cuando se lee un único bloque, la velocidad de transferencia de datos es igual que en un disco, pero los restantes n - 1 discos están libres de realizar cualquier otra acción.

La distribución en el nivel de bloque es la forma de distribución de datos más usada. También son posibles otros niveles de distribución, como los bytes de cada sector o los sectores de cada bloque.

En resumen, hay dos objetivos principales para el paralelismo en un sistema de discos:

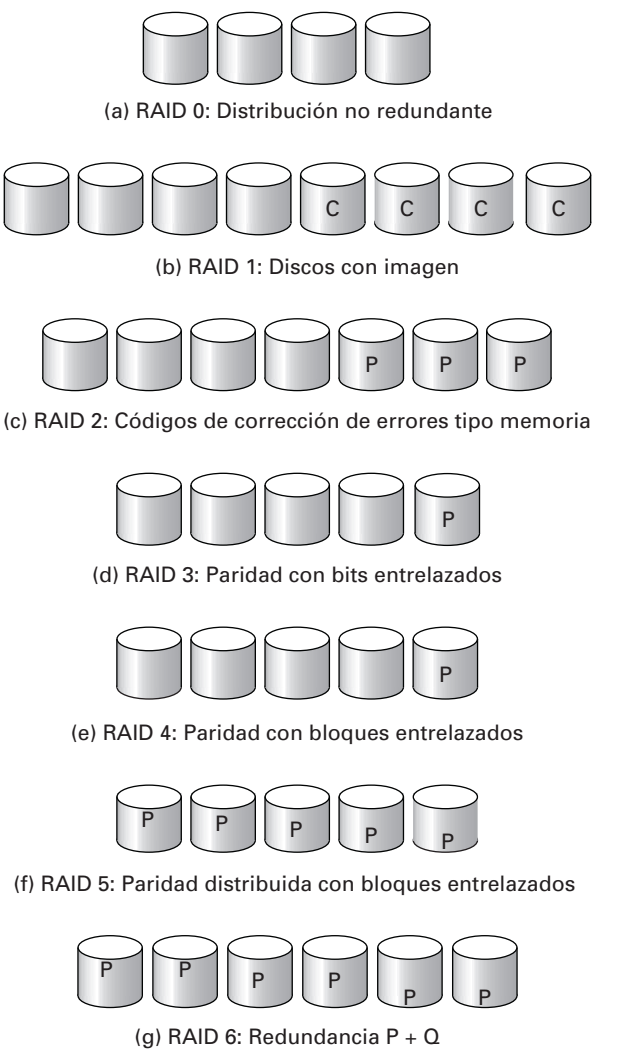
Equilibrar la carga de varios accesos de pequeño tamaño (accesos a bloque) de manera que la productividad de ese tipo de accesos aumente.

Convertir en paralelos los accesos de gran tamaño para que su tiempo de respuesta se reduzca.

11.3.3. Niveles de RAID

La creación de imágenes proporciona gran fiabilidad pero resulta costosa. La distribución proporciona velocidades de transferencia de datos elevadas pero no mejora la fiabilidad. Se han propuesto esquemas alternativos para proporcionar redundancia a bajo costo utilizando la idea de la distribución de los discos combinada con los bits de «paridad» (que se describen a continuación). Estos esquemas tienen diferentes compromisos entre el coste y el rendimiento. Los esquemas se clasifican en niveles denominados **niveles de RAID**, como se muestra en la Figura 11.4 (en la figura, *P* indica los bits para la corrección de errores y *C* indica una segunda copia de los datos). En todos los casos descritos en la figura se guardan cuatro discos de datos y los discos adicionales para guardar la información redundante para la recuperación en caso de fallo.

- **RAID de nivel 0** se refiere a disposiciones de discos con distribución en el nivel de bloque pero sin redundancia (como la creación de imágenes o los bits de paridad). En la Figura 11.4a se muestra una disposición de tamaño cuatro.
- **RAID de nivel 1** se refiere a la creación de imágenes del disco con distribución de bloques. En la Figura 11.4b se muestra una organización con imagen que tiene cuatro discos con datos.
- **RAID de nivel 2** también se conoce como organización de códigos de corrección de errores tipo memoria (*memory-style error-correcting-code organization*, ECC). Los sistemas de memoria hace tiempo que realizan la detección de errores utilizando los bits de paridad. Cada byte del sistema de memoria puede tener asociado un bit de paridad que registra si el número de bits del byte que valen uno es par



(a) RAID 0: Distribución no redundante

(b) RAID 1: Discos con imagen

(c) RAID 2: Códigos de corrección de errores tipo memoria

(d) RAID 3: Paridad con bits entrelazados

(e) RAID 4: Paridad con bloques entrelazados

(f) RAID 5: Paridad distribuida con bloques entrelazados

(g) RAID 6: Redundancia P + Q

FIGURA 11.4. Niveles de RAID.

(paridad = 0) o impar (paridad = 1). Si uno de los bits del byte se deteriora (un uno se transforma en cero o viceversa) la paridad del byte se modifica y, por tanto, no coincidirá con la paridad guardada. De manera parecida, si el bit de paridad guardado se deteriora no coincidirá con la paridad calculada. Por tanto, todos los errores de un bit son detectados por el sistema de memoria. Los esquemas de corrección de errores guardan dos o más bits adicionales y pueden reconstruir los datos si se deteriora un solo bit.

La idea de los códigos para la corrección de errores puede utilizarse directamente en los conjuntos de discos mediante la distribución de los bytes entre los discos. Por ejemplo, el primer bit de cada byte puede guardarse en el disco uno, el segundo en el disco dos, etcétera, hasta que se guarde el octavo bit en el disco ocho y los bits para la corrección de errores se guardan en los demás discos.

Este esquema se muestra gráficamente en la Figura 11.4. Los discos marcados como *P* guardan los bits para la corrección de errores. Si uno de los discos falla, los bits restantes del byte y los bits para la corrección de errores asociados pueden leerse de los demás discos y utilizarse para reconstruir los datos deteriorados. En la Figura 11.4c se muestra una disposición de tamaño cuatro; obsérvese que RAID de nivel 2 sólo necesita la sobrecarga de tres discos para cuatro discos de datos, a diferencia de RAID de nivel 1, que necesitaba la sobrecarga de cuatro discos.

- **RAID de nivel 3**, u organización de paridad con bits entrelazados, mejora respecto al nivel 2 destacando que, a diferencia de los sistemas de memoria, los controladores de disco pueden detectar si un sector se ha leído correctamente, por lo que se puede utilizar un solo bit de paridad para la corrección y la detección de los errores. La idea es la siguiente. Si uno de los sectores se deteriora, se sabe exactamente el sector que es y para cada bit del mismo se puede determinar si es un uno o un cero calculando la paridad de los bits correspondientes a partir de los sectores de los demás discos. Si la paridad de los bits restantes es igual que la paridad guardada, el bit ausente es un cero; en caso contrario, es un uno.

  RAID de nivel 3 es tan bueno como el nivel 2, pero resulta menos costoso en cuanto al número de discos adicionales (sólo tiene la sobrecarga de un disco), por lo que el nivel 2 no se utiliza en la práctica. Este esquema se muestra gráficamente en la Figura 11.4d.

  RAID de nivel 3 tiene dos ventajas respecto al nivel 1. Sólo se necesita un disco de paridad para varios discos normales, en comparación con un disco imagen por cada disco en el nivel 1, por lo que se reduce la sobrecarga de almacenamiento. Dado que los procesos de lectura y de escritura de un byte se extienden en varios discos, con la distribución de datos en *N* vías, la velocidad de transferencia es *N* veces tan rápida como con un solo disco. Por otro lado, RAID de nivel 3 permite un menor número de operaciones de E/S por segundo, dado que todos los discos tienen que participar en cada solicitud de E/S.

- **RAID de nivel 4**, u organización de paridad con bloques entrelazados, usa distribución de bloques, y además guarda un bloque de paridad en un disco aparte para los bloques correspondientes de los otros *N* discos. Este esquema se muestra gráficamente en la Figura 11.4e. Si uno de los discos falla puede utilizarse el bloque de paridad con los bloques correspondientes de los demás discos para restaurar los bloques del disco averiado.

  La lectura de un bloque sólo accede a un disco, lo que permite que los demás discos procesen otras solicitudes. Por tanto, la velocidad de transferencia de datos de cada acceso es menor, pero se pueden ejecutar en paralelo varios accesos de lectura, lo que produce una mayor velocidad global de E/S. Las velocidades de transferencia para los procesos de lectura de gran tamaño son elevadas, dado que se pueden leer todos los discos en paralelo; los procesos de escritura de gran tamaño también tienen velocidades de transferencia elevadas, dado que los datos y la paridad pueden escribirse en paralelo.

  Los procesos de escritura independientes de pequeño tamaño, por otro lado, no pueden realizarse en paralelo. La escritura de un bloque tiene que tener acceso al disco en el que se guarda ese bloque, así como al disco de paridad, dado que hay que actualizar el bloque de paridad. Además, hay que leer tanto el valor anterior del bloque de paridad como el del bloque que se escribe para calcular la nueva paridad. Por tanto, un solo proceso de escritura necesita cuatro accesos a disco: dos para leer los dos bloques antiguos y otros dos para escribir los dos nuevos.

- **RAID de nivel 5**, o paridad distribuida con bloques entrelazados, mejora respecto al nivel 4 dividiendo los datos y la paridad entre los $N + 1$ discos en vez de guardar los datos en *N* discos y la paridad en uno. En el nivel 5 todos los discos pueden participar en la atención a las solicitudes de lectura, a diferencia de RAID de nivel 4, en que el disco de paridad no puede participar, por lo que el nivel 5 aumenta el número total de solicitudes que pueden atenderse en una cantidad de tiempo dada. Para cada conjunto de *N* bloques lógicos, uno de los discos guarda la paridad y los otros *N* guardan los bloques.

  Esta configuración se muestra gráficamente en la Figura 11.4f, en la que las *P* están distribuidas entre todos los discos. Por ejemplo, con una disposición de cinco discos el bloque de paridad, marcado como Pk para los bloques lógicos $4k$, $4k+1$,

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

$4k$+2, $4k$+3 se guardan en el disco (k mod 5) + 1; los bloques correspondientes de los otros cuatro discos guardan los cuatro bloques de datos $4k$ a $4k$+3. La siguiente tabla muestra cómo se disponen los primeros veinte bloques, numerados de 0 a 19, y sus bloques de paridad. El patrón mostrado se repite para los siguientes bloques.

Obsérvese que un bloque de paridad no puede guardar la paridad de los bloques del mismo disco, dado que entonces un fallo del disco supondría la pérdida de datos además de la de la paridad y, por tanto, no sería recuperable. El nivel 5 incluye al nivel 4, dado que ofrece mejor rendimiento de lectura y de escritura por el mismo coste, por lo que el nivel 4 no se utiliza en la práctica.

- **RAID de nivel 6**, también denominado esquema de redundancia P+Q, es muy parecido a RAID de nivel 5 pero guarda información redundante adicional para protección contra fallos de disco múltiples. En lugar de utilizar la paridad se utilizan códigos para la corrección de errores como los de Reed-Solomon (véanse las notas bibliográficas). En el esquema mostrado en la Figura 11.4g se guardan dos bits de datos redundantes por cada cuatro bits de datos (en comparación con un bit de paridad del nivel 5) y el sistema puede tolerar dos fallos del disco.

Finalmente, se debe destacar que se han propuesto varias alternativas a los esquemas RAID básicos aquí descritos.

Algunos fabricantes usan su propia terminología para describir sus implementaciones RAID[2]. Sin embargo, la terminología que se ha presentado es la más ampliamente usada.

11.3.4. Elección del nivel RAID adecuado

Los factores a tener en cuenta al elegir un nivel RAID son:

- Costo económico extra de los requisitos de almacenamiento en disco.
- Requisitos de rendimiento en términos del número de operaciones E/S.
- Rendimiento cuando falla un disco.
- Rendimiento durante la reconstrucción (esto es, mientras los datos del disco averiado se reconstruyen en un nuevo disco).

Si un disco falla, el tiempo que se tarda en reconstruir los datos que contenía puede ser significativo, y variará con el nivel RAID utilizado. La reconstrucción resulta más sencilla para RAID de nivel 1, dado que los datos pueden copiarse de otro disco; para los otros niveles hay que tener acceso a todos los demás discos de la disposición para reconstruir los datos del disco averiado. El **rendimiento en la reconstrucción** de un sistema RAID puede ser un factor importante si se necesita un aporte continuo de datos, como ocurre en los sistemas de bases de datos de alto rendimiento. Además, dado que el tiempo de reconstrucción puede formar parte del tiempo de reparación, el rendimiento de la reconstrucción influye en el tiempo medio entre fallos.

RAID de nivel 0 se usa en aplicaciones de alto rendimiento donde la seguridad de los datos no es crítica. Dado que los niveles 2 y 4 de RAID se incluyen en los niveles 3 y 5 de RAID, la elección de los niveles RAID se limita a los niveles restantes. La distribución de bits (nivel 3) se usa raramente, dado que la distribución de bloques (nivel 5) da buenas velocidades de transferencia de datos para grandes transferencias. Para pequeñas transferencias, el tiempo de acceso a disco es el factor dominante, así que el beneficio de las lecturas paralelas disminuye. De hecho, el nivel 3 puede funcionar peor que el nivel 5 para una pequeña transferencia, ya que la transferencia sólo se completa cuando los sectores correspondientes en todos los discos se hayan encontrado; la latencia media de la disposición de discos se comporta de forma muy parecida a la latencia en el caso peor para un único disco, descartando los beneficios de las mayores velocidades de transferencia. El nivel 6 no se soporta actualmente en muchas implementaciones RAID, pero ofrece una mejor fiabilidad que el nivel 5 y se puede usar en aplicaciones donde la seguridad de datos es muy importante.

La elección entre RAID de nivel 1 y de nivel 5 es más difícil de tomar. RAID de nivel 1 es popular para las aplicaciones como el almacenamiento de archivos de registro histórico en un sistema de bases de datos, ya que ofrece el mejor rendimiento en escritura. RAID de nivel 5 tiene una menor sobrecarga de almacenamiento que el nivel 1, pero tiene una mayor sobrecarga en las escrituras. Para las aplicaciones donde los datos se leen frecuentemente y se escriben raramente, el nivel 5 es la elección adecuada.

Las capacidades de almacenamiento en disco han estado aumentando a una velocidad sobre el 50 por ciento al año durante muchos años, y el costo por byte

[2] Por ejemplo, algunos productos usan el nivel 1 de RAID para referirse a discos imagen sin distribución y el nivel 1 + 0 o 10 para los discos imagen con distribución. Esta distinción no es realmente necesaria, ya que la no distribución se puede ver como un caso especial de distribución, en concreto, la distribución sobre un único disco.

ha estado decreciendo a la misma velocidad. Como resultado, para muchas aplicaciones existentes de bases de datos con requisitos moderados de almacenamiento, el costo económico del almacenamiento extra en disco necesario para la creación de imágenes ha sido relativamente pequeño (sin embargo, el costo económico extra, sigue siendo un aspecto significativo para las aplicaciones de almacenamiento intensivo como el almacenamiento de datos de vídeo). Las velocidades de acceso se han mejorado a una velocidad mucho menor (cerca de un factor 3 durante 10 años), mientras que el número de operaciones E/S requeridas por segundo se han incrementado enormemente, particularmente en los servidores de aplicaciones Web.

El nivel 5 de RAID, que incrementa el número de operaciones E/S necesarias para escribir un único bloque lógico, sufre una penalización de tiempo significativa en términos del rendimiento en escritura. El nivel 1 de RAID es, por tanto, la elección adecuada para muchas aplicaciones con requisitos moderados de almacenamiento y altos requisitos de E/S.

Los diseñadores de sistemas RAID tienen también que hacer otras decisiones. Por ejemplo, la cantidad de discos que habrá en la disposición y los bits que debe proteger cada bit de paridad. Si hay más discos en la disposición, las velocidades de transferencia de datos son mayores pero el sistema sería más caro. Si hay más bits protegidos por cada bit de paridad, la sobrecarga de espacio debida a los bits de paridad es menor, pero hay más posibilidades de que falle un segundo disco antes de que el primer disco en averiarse esté reparado y que eso dé lugar a la pérdida de datos.

11.3.5. Aspectos hardware

Otro aspecto en la elección de implementaciones RAID se encuentra en el nivel hardware. RAID se puede implementar sin cambios en el nivel hardware modificando sólo el software. Tales implementaciones se conocen como **RAID software**. Sin embargo, hay beneficios significativos al construir hardware de propósito especial para dar soporte a RAID, que se describen a continuación; los sistemas con soporte hardware especial se denominan sistemas **RAID hardware**.

Las implementaciones RAID hardware pueden usar RAM no volátil para registrar las escrituras que es necesario ejecutar; en caso de fallo de corriente antes de que se complete una escritura, el sistema se restaura recuperando información acerca de las escrituras incompletas de la memoria RAM no volátil y entonces completa las escrituras. Sin el soporte hardware, se necesita trabajo extra para detectar los bloques que se hayan escrito parcialmente antes del fallo de corriente (véase el Ejercicio 11.4).

Algunas implementaciones RAID permiten el **intercambio en caliente**; esto es, los discos averiados se pueden eliminar y reemplazar por otros nuevos sin desconectar la corriente. El intercambio en caliente reduce el tiempo medio de reparación, ya que el cambio de un disco no debe esperar hasta que se pueda apagar el sistema. De hecho, muchos sistemas críticos actuales se ejecutan con una planificación 24 × 7; esto es, se ejecutan 24 horas al día y 7 días a la semana, sin proporcionar ningún momento para apagar el sistema y cambiar el disco averiado. Como resultado, el tiempo medio de reparación se reduce en gran medida, minimizando la posibilidad de pérdida de datos. El disco averiado se puede reemplazar en los ratos libres.

La fuente de alimentación, o el controlador de disco, o incluso la interconexión del sistema en un sistema RAID podría llegar a ser un punto de fallo que detendría el funcionamiento del sistema RAID. Para evitar esta posibilidad, las buenas implementaciones RAID tienen varias fuentes de alimentación (con baterías de respaldo que les permiten continuar funcionando aunque se corte la corriente). Tales sistemas RAID tienen varios controladores de disco y varias interconexiones para conectarlos con el sistema informático (o a la red de los sistemas informáticos). Así, el fallo de cualquier componente no detendrá el funcionamiento del sistema RAID.

11.3.6. Otras aplicaciones de RAID

Los conceptos de RAID se han generalizado a otros dispositivos de almacenamiento, incluyendo los conjuntos de cintas, e incluso a la transmisión de datos por sistemas de radio. Cuando se aplican a los conjuntos de cintas, las estructuras RAID pueden recuperar datos aunque se deteriore una de las cintas de la disposición. Cuando se aplican a la transmisión de datos, cada bloque de datos se divide en unidades menores y se transmite junto con una unidad de paridad; si por algún motivo no se recibe alguna de las unidades, se puede reconstruir a partir del resto.

11.4. ALMACENAMIENTO TERCIARIO

En un sistema de bases de datos de gran tamaño puede que parte de los datos tenga que residir en almacenamiento terciario. Los dos medios de almacenamiento terciario más frecuentes son los discos ópticos y las cintas magnéticas.

11.4.1. Discos ópticos

Los discos compactos son un medio popular de distribución de software, datos multimedia como el sonido y las imágenes, y otra información editada de manera

electrónica. Tienen una elevada capacidad de almacenamiento (640 megabytes) y resulta barato producir masivamente los discos.

Los discos de vídeo digital (DVD, Digital Video Disk) están reemplazando a los discos compactos en las aplicaciones que requieren grandes cantidades de datos. Los discos en formato DVD-5 pueden almacenar 4,7 gigabytes de datos (en una superficie de grabación), mientras que los discos en formato DVD-9 pueden almacenar 8,5 gigabytes de datos (en dos superficies de disco). La grabación en ambas caras de un disco ofrece incluso mayores capacidades; los formatos DVD-10 y DVD-18, que son las versiones de doble cara de DVD-5 y DVD-9, pueden almacenar respectivamente 9,4 y 17 gigabytes.

Las unidades CD y DVD presentan tiempos de búsqueda mucho mayores (100 milisegundos es un valor frecuente) que las unidades de discos magnéticos, debido a que el dispositivo de cabezas es más pesado. Las velocidades rotacionales son generalmente menores, aunque las unidades CD y DVD más rápidas tienen velocidades rotacionales alrededor de 3.000 revoluciones por minuto, que son comparables a las velocidades de los discos magnéticos de gama baja. Las velocidades rotacionales se correspondían inicialmente con las normas de los CD de sonido, y las velocidades de las unidades DVD se correspondían inicialmente con las normas de los DVD de vídeo, pero hoy en día se dispone de unidades que hacen girar los discos muchas veces la velocidad normal.

Las velocidades de transferencia de datos son algo menores que los discos magnéticos. Las unidades CD actuales leen entre 3 y 6 megabytes por segundo, y las unidades DVD actuales leen de 8 a 15 megabytes por segundo. Al igual que las unidades de discos magnéticos, los discos ópticos almacenan más datos en las pistas exteriores y menos en las interiores. La velocidad de transferencia de las unidades ópticas se caracteriza por $n\times$, que significa que la unidad soporta transferencias n veces la velocidad normal; las velocidades 50× para CD y 12× para DVD son comunes actualmente.

Las versiones de escritura única de los discos ópticos (CD-R y DVD-R) son populares para la distribución de datos y en particular para el almacenamiento de archivo de datos porque tienen una gran capacidad, un tiempo de vida mayor que los discos magnéticos y pueden retirarse de la unidad y almacenarse en un lugar remoto. Dado que no pueden sobrescribirse, se pueden usar para almacenar información que no debería cambiarse, como los rastros de auditoría. Las versiones de varias escrituras (CR-RW, DVD-RW y DVD-RAM) también se usan para archivo.

Los **cambiadores de discos** son dispositivos que guardan un gran número de discos ópticos (hasta varios cientos) y los cargan automáticamente bajo demanda en una de las pocas unidades (usualmente entre una y diez). La capacidad de almacenamiento agregada de tal sistema puede tener muchos terabytes. Cuando se accede a un disco, se carga mediante un brazo mecánico desde una estantería en la unidad (cualquier disco que estuviese previamente en la unidad se debe devolver a la estantería). El tiempo de carga y descarga suele ser del orden de segundos (mucho más lento que los tiempos de acceso a disco).

11.4.2. Cintas magnéticas

Aunque las cintas magnéticas son relativamente permanentes y pueden albergar grandes volúmenes de datos, resultan lentas en comparación con los discos magnéticos y ópticos. Aún más importante, la cinta magnética está limitada al acceso secuencial. Por tanto, resulta inadecuada para proporcionar el acceso aleatorio que cumple la mayor parte de los requisitos del almacenamiento secundario, aunque históricamente, antes del uso de los discos magnéticos, las cintas se usaron como medio de almacenamiento secundario.

Las cintas se utilizan principalmente para copias de seguridad, para el almacenamiento de la información poco utilizada y como medio sin conexión para transferir información de un sistema a otro. Las cintas también se usan para almacenar grandes volúmenes de datos, tales como datos vídeo o de imagen que, o no es necesario acceder rápidamente a ellos o son tan voluminosos que el almacenamiento en disco sería muy caro.

Una cinta se guarda en una bobina y se enrolla o desenrolla sobre una cabeza de lectura y escritura. El desplazamiento hasta el punto correcto de la cinta puede tardar minutos en vez de milisegundos; una vez en posición, sin embargo, las unidades de cinta pueden escribir los datos con densidades y velocidades que se aproximan a las de las unidades de disco. Las capacidades varían en función de la longitud y de la anchura de la cinta y de la densidad con la que la cabeza pueda leer y escribir. El mercado está actualmente dividido en una amplia variedad de formatos de cinta. Las capacidades actuales disponibles varían de unos pocos gigabytes (con el formato **DAT [Digital Audio Tape, cinta de audio digital**]), 10 a 40 gigabytes (con el formato **DLT [Digital Linear Tape, cinta lineal digital**]), 100 gigabytes y aún más (con el formato **Ultrium**), hasta 330 gigabytes (con los formatos de cinta de **exploración helicoidal de Ampex**). Las velocidades de transferencia de datos son del orden de algunos hasta decenas de megabytes por segundo.

Las unidades de cinta son muy fiables y los buenos sistemas de unidades de cinta realizan la lectura de los datos recién escritos para asegurar que se han registrado correctamente. Sin embargo, las cintas presentan límites en cuanto al número de veces que se pueden leer o escribir con fiabilidad.

Algunos formatos de cinta (como el formato **Accelis**) soportan velocidades de búsqueda mayores (del orden de decenas de segundos), lo cual es importante para las aplicaciones que necesitan un rápido acceso a muy grandes cantidades de datos, mayores de lo que

económicamente podría caber en una unidad de disco. La mayoría del resto de formatos de cinta proporcionan mayores capacidades al precio de un acceso más lento; estos formatos son ideales para la copia de seguridad de datos, donde las búsquedas rápidas no son importantes.

Los *cambiadores de cintas*, al igual que los cambiadores de discos ópticos, guardan gran número de cintas con unas cuantas unidades en las que se pueden montar las cintas; se utilizan para guardar grandes volúmenes de datos, que pueden llegar a varios terabytes (10^{12} bytes) con tiempos de acceso del orden de segundos hasta unos cuantos minutos. Las aplicaciones que necesitan estos enormes almacenamientos de datos incluyen los sistemas de imágenes que reúnen los datos de los satélites de teledetección, y grandes bibliotecas de vídeo para las cadenas de televisión.

11.5. ACCESO AL ALMACENAMIENTO

Las bases de datos se corresponden con cierto número de archivos diferentes que mantiene el sistema operativo subyacente. Estos archivos residen permanentemente en los discos, con copias de seguridad en cinta. Cada archivo se divide en unidades de almacenamiento de longitud constante denominadas **bloques**, que son las unidades de asignación de almacenamiento y de transferencia de datos. En el Apartado 11.6 se discutirán varias maneras de organizar los datos en archivos de manera lógica.

Cada bloque puede contener varios elementos de datos. El conjunto concreto de elementos de datos que contiene cada bloque viene determinado por la forma de organización física de los datos que se utilice (véase el Apartado 11.6). Se supondrá que ningún elemento de datos ocupa dos o más bloques. Esta suposición es realista para la mayor parte de las aplicaciones de procesamiento de datos, como el ejemplo bancario aquí propuesto.

Uno de los principales objetivos del sistema de bases de datos es minimizar el número de transferencias de bloques entre el disco y la memoria. Una manera de reducir el número de accesos al disco es mantener en la memoria principal todos los bloques que sea posible. El objetivo es maximizar la posibilidad de que, cuando se tenga acceso a un bloque, ya se encuentre en la memoria principal y, por tanto, no se necesite realizar un acceso al disco.

Dado que no resulta posible mantener en la memoria principal todos los bloques hay que gestionar la asignación del espacio disponible en la memoria principal para el almacenamiento de los mismos. La **memoria intermedia** (buffer) es la parte de la memoria principal disponible para el almacenamiento de las copias de los bloques del disco. Siempre se guarda en el disco una copia de cada bloque, pero esta copia puede ser una versión del bloque más antigua que la versión de la memoria intermedia. El subsistema responsable de la asignación del espacio de la memoria intermedia se denomina **gestor de la memoria intermedia**.

11.5.1. Gestor de la memoria intermedia

Los programas de un sistema de bases de datos formulan solicitudes (es decir, llamadas) al gestor de la memoria intermedia cuando necesitan un bloque del disco. Si el bloque ya se encuentra en la memoria intermedia se pasa al solicitante la dirección del bloque en la memoria principal. Si el bloque no se halla en la memoria intermedia, el gestor de la memoria intermedia asigna en primer lugar espacio al bloque en la memoria intermedia, descartando algún otro bloque si hace falta para hacer sitio para el nuevo bloque. Sólo se vuelve a escribir en el disco el bloque que se descarta si se modificó desde la última vez que se escribió en el disco. A continuación, el gestor de la memoria intermedia lee el bloque del disco y lo escribe en la memoria intermedia, y pasa la dirección del bloque en la memoria principal al solicitante. Las acciones internas del gestor de la memoria intermedia resultan transparentes para los programas que formulan solicitudes de bloques de disco.

Si se está familiarizado con los conceptos de los sistemas operativos se observará que el gestor de la memoria intermedia no parece ser más que un gestor de la memoria virtual, como los que se hallan en la mayor parte de los sistemas operativos. Una diferencia estriba en que el tamaño de la base de datos puede ser mucho mayor que el espacio de direcciones de hardware de la máquina, por lo que las direcciones de memoria no resultan suficientes para direccionar todos los bloques de memoria. Además, para dar un buen servicio al sistema de bases de datos el gestor de la memoria intermedia debe utilizar técnicas más complejas que los esquemas de gestión de la memoria virtual habituales:

- **Estrategia de sustitución.** Cuando no queda espacio libre en la memoria intermedia hay que eliminar un bloque de ésta antes de que se pueda escribir en él otro nuevo. Generalmente los sistemas operativos utilizan un esquema **menos recientemente utilizado** (Least Recently Used, LRU), en el que se vuelve a escribir en el disco y se elimina de la memoria intermedia el bloque al que se ha hecho referencia menos recientemente.
- **Bloques clavados.** Para que el sistema de bases de datos pueda recuperarse de las caídas (Capítulo 17) resulta necesario limitar las ocasiones en que se puede volver a escribir el bloque en el dis-

co. Se dice que un bloque al que no se le permite que se vuelva a escribir en el disco está **clavado**. Aunque muchos sistemas operativos no permiten trabajar con bloques clavados, esta prestación resulta esencial para la implementación de un sistema de bases de datos resistente a las caídas.

- **Salida forzada de los bloques.** Hay situaciones en las que resulta necesario volver a escribir el bloque en el disco, aunque no se necesite el espacio de memoria intermedia que ocupa. Este proceso de escritura se denomina **salida forzada** del bloque. Se verá el motivo de que se necesite la salida forzada en el Capítulo 17; para resumirlo, el contenido de la memoria principal y, por tanto, el de la memoria intermedia se pierde en las caídas, mientras que los datos del disco suelen sobrevivir a ellos.

11.5.2. Políticas para la sustitución de la memoria intermedia

El objetivo de las estrategias de sustitución de los bloques de la memoria intermedia es la minimización de los accesos al disco. En los programas de propósito general no resulta posible predecir con precisión los bloques a los que se hará referencia. Por tanto, los sistemas operativos utilizan la pauta anterior de las referencias a los bloques como forma de predecir las referencias futuras. La suposición que suele hacerse es que es probable que se vuelva a hacer referencia a los bloques a los que se ha hecho referencia recientemente. Por tanto, si hay que sustituir un bloque, se sustituye el bloque al que se ha hecho referencia menos recientemente. Este enfoque se denomina **esquema de sustitución de bloques LRU**.

LRU es un esquema de sustitución aceptable para los sistemas operativos. Sin embargo, los sistemas de bases de datos pueden predecir la pauta de las referencias futuras con más precisión que los sistemas operativos. Las peticiones del usuario al sistema de bases de datos comprenden varias etapas. El sistema de bases de datos suele poder determinar con antelación los bloques que se necesitarán examinando cada una de las etapas necesarias para llevar a cabo la operación solicitada por el usuario. Por tanto, a diferencia de los sistemas operativos, que deben confiar en el pasado para predecir el futuro, los sistemas de bases de datos pueden tener información concerniente al menos al futuro a corto plazo.

Para ilustrar la manera en que la información relativa al futuro acceso a los bloques permite mejorar la estrategia LRU considérese el procesamiento de la expresión del álgebra relacional

$$prestatario \bowtie cliente$$

Supóngase que la estrategia escogida para procesar esta solicitud viene dada por el programa en seudocódigo mostrado en la Figura 11.5 (se estudiarán otras estrategias en el Capítulo 13).

Supóngase que las dos relaciones de este ejemplo se guardan en archivos diferentes. En este ejemplo se puede ver que, una vez que se haya procesado la tupla de *prestatario*, no se vuelve a necesitar. Por tanto, una vez que se ha completado el procesamiento de un bloque completo de tuplas de *prestatario*, ese bloque ya no se necesita en la memoria principal, aunque se haya utilizado recientemente. Deben darse instrucciones al gestor de la memoria intermedia para liberar el espacio ocupado por el bloque de *prestatario* tan pronto como se haya procesado la última tupla. Esta estrategia de gestión de la memoria intermedia se denomina estrategia *de* **extracción inmediata**.

Considérense ahora los bloques que contienen las tuplas de *cliente*. Hay que examinar una vez cada bloque de las tuplas *cliente* por cada tupla de la relación *prestatario*. Cuando se completa el procesamiento del bloque *cliente* se sabe que no se tendrá nuevamente acceso a este hasta que se hayan procesado todos los demás bloques de *cliente*. Por tanto, el bloque de *cliente* al que se haya hecho referencia más recientemente será el último bloque al que se vuelva a referenciar, y el bloque de *cliente* al que se haya hecho referencia menos recientemente será el bloque al que se vuelva a hacer referencia a continuación. Este conjunto de suposiciones es el reverso exacto del que forma la base de la estrategia LRU. En realidad, la estrategia óptima para la sustitución de bloques es la **estrategia más recientemente**

```
for each tupla p de prestatario do
    for each tupla c de cliente do
        if p[nombre-cliente] = c[nombre-cliente]
        then begin
            sea x una tupla definida de la manera siguiente:
            x[nombre-cliente] := p[nombre-cliente]
            x[número-préstamo] := p[número-préstamo]
            x[calle-cliente] := c[calle-cliente]
            x[ciudad-cliente] := c[ciudad-cliente]
            incluir la tupla x como parte del resultado de prestatario ⋈ cliente
        end

    end
end
```

FIGURA 11.5. Procedimiento para calcular la reunión.

utilizada (**Most Recently Used**, **MRU**). Si hay que eliminar de la memoria intermedia un bloque de *cliente*, la estrategia MRU escoge el bloque utilizado más recientemente.

Para que la estrategia MRU funcione correctamente en el ejemplo propuesto, el sistema debe clavar el bloque de *cliente* que se esté procesando. Después de que se haya procesado la última tupla de *cliente* el bloque se desclava y se transforma en el bloque utilizado más recientemente.

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 concerniente a la probabilidad de que una solicitud haga referencia a una relación concreta. Por ejemplo, el diccionario de datos (como se verá en detalle en el Apartado 11.8) que guarda el esquema lógico de las relaciones y su información del 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 por otros factores. En el Capítulo 12 se discuten los índices de los archivos. Dado que puede que se tenga acceso más frecuentemente al índice de un archivo que al propio archivo, el gestor de la memoria intermedia no deberá, en general, eliminar los bloques del índice de la memoria principal si se dispone de alternativas.

La estrategia ideal para la sustitución de bloques necesita información sobre las operaciones de la bases de datos (las que se estén realizando y las que se realizarán en el futuro). No se conoce ninguna estrategia aislada que responda bien a todas las situaciones posibles. En realidad, un número sorprendentemente grande de bases de datos utilizan LRU a pesar de los defectos de esa estrategia. 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 al bloque. Si el sistema está procesando de manera concurrente las solicitudes de varios usuarios, puede que el subsistema para el control de la 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 la concurrencia que indique las solicitudes que se posponen, puede utilizar esta información para modificar su estrategia de sustitución de los bloques. De manera específica, los bloques que necesiten las solicitudes activas (no pospuestas) pueden retenerse en la memoria intermedia a expensas de los bloques que necesiten las solicitudes pospuestas.

El subsistema para la recuperación de caídas (Capítulo 17) impone restricciones estrictas a la sustitución de los bloques. Si se ha modificado un bloque, no se permite al gestor de la memoria intermedia volver a copiar la versión nueva del bloque de la memoria intermedia al disco, dado que eso destruiría la versión anterior. Por el contrario, el gestor de bloques debe solicitar permiso del subsistema para la recuperación de averías antes de escribir los bloques. Puede que el subsistema para la recuperación de averías exija que se fuerce la salida de otros bloques antes de conceder autorización al gestor de la memoria intermedia para escribir 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 averías.

11.6. ORGANIZACIÓN DE LOS ARCHIVOS

Los **archivos** se organizan lógicamente como secuencias de registros. Estos registros se corresponden con los bloques del disco. Los archivos se proporcionan como un instrumento fundamental de los sistemas operativos, por lo que se supondrá 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 archivos.

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

Un enfoque de la correspondencia entre la base de datos y los archivos es utilizar varios y guardar los registros de cada una de las diferentes longitudes fijas existentes en cada uno de esos archivos. Los archivos con registros de longitud fija son más sencillos de implementar que los archivos con registros de longitud variable. Muchas de las técnicas utilizadas para los primeros pueden aplicarse al caso de longitud variable. Por tanto, se comienza considerando un archivo 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 de la manera siguiente:

```
type depósito = record
                    número-cuenta: char(10);
                    nombre-sucursal: char (22);
                    saldo: real;
end
```

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 11.6. Archivo que contiene los registros de *cuenta*.

Si se supone que cada carácter ocupa un byte y que un valor de tipo real ocupa 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 registro, etcétera (Figura 11.6). Sin embargo, hay dos problemas con este enfoque sencillo:

1. Resulta difícil borrar un registro de esta estructura. Se debe rellenar el espacio ocupado por el registro que hay que borrar con algún otro registro del archivo o tener algún medio de marcar los registros borrados para que puedan pasarse por alto.
2. A menos que el tamaño de los bloques sea un múltiplo de cuarenta (lo que resulta improbable) algunos de los registros se saltarán 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 ese tipo de registros.

Cuando se borra un registro se puede desplazar el registro situado a continuación al espacio ocupado anteriormente por el registro borrado y hacer lo mismo con los demás registros hasta que todos los registros situados a continuación del borrado se hayan desplazado hacia delante (Figura 11.7). Un enfoque de este tipo necesita desplazar gran número de registros. Resultaría más sencillo desplazar simplemente el último registro del archivo al espacio ocupado por el registro borrado (Figura 11.8).

No resulta deseable desplazar los registros para ocupar el espacio liberado por los registros borrados, dado

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 11.7. El archivo de la Figura 11.6 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 11.8. El archivo de la Figura 11.6 con el registro 2 borrado y el último registro desplazado.

que se necesitan accesos adicionales a los bloques. Dado que las inserciones tienden a ser más frecuentes que los borrados, sí resulta aceptable dejar libre el espacio 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, dado que resulta difícil 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 contendrá gran variedad de información sobre el archivo. Por ahora, todo lo que hace falta guardar ahí 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**. En la Figura 11.9 se muestra el archivo de la Figura 11.6 después de haberse borrado los registros 1, 4 y 6.

Al insertar un registro nuevo se utiliza el registro indicado por la cabecera. Se cambia el puntero de la cabecera para que apunte 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

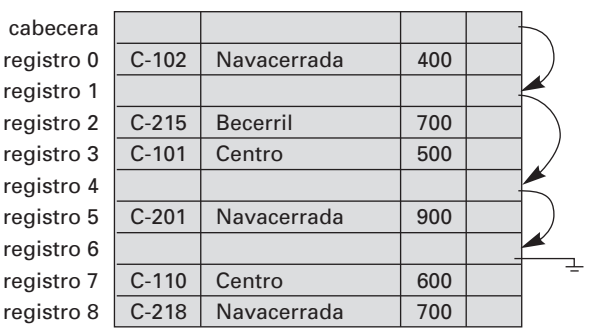


FIGURA 11.9. El archivo de la Figura 11.6 después del borrado de los registros 1, 4 y 6.

espacio que deja libre el 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 puede que sólo llene parte del mismo.

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
- Tipos de registro que permiten longitudes variables para uno o varios de los campos
- Tipos de registro que permiten campos repetidos

Existen diferentes técnicas para implementar los registros de longitud variable. Con fines ilustrativos se utilizará un ejemplo para mostrar las diversas técnicas de implementación. Se tomará en consideración una representación diferente de la información de *cuenta* guardada en el archivo de la Figura 11.6, en la que se utiliza un registro de longitud variable para el nombre de cada sucursal y para toda la información de las cuentas de cada sucursal. El formato del registro es

```
type lista-cuentas = record
        nombre-sucursal : char (22);
        información-cuenta : array [1 .. ∞] of record;
                        número-cuenta : char(10);
                        saldo : real;
                                                end
                    end
```

Se define *información-cuenta* como un *array* con un número arbitrario de elementos, por lo que no hay ningún límite para el tamaño que pueden tener los registros (hasta el tamaño del disco, ¡por supuesto!).

11.6.2.1. Representación en cadenas de bytes

Un método sencillo de implementar los registros de longitud variable es adjuntar un símbolo especial de *fin-de-registro* (⊥) al final de cada registro. Así se puede guardar cada registro como una cadena de bytes consecutivos. En la Figura 11.10 se muestra una organización de este tipo que representa el archivo con registros de longitud fija de la Figura 11.6 utilizando registros de longitud variable. Una versión alternativa de la representación en cadenas de bytes guarda la longitud del registro al comienzo de cada registro en lugar de utilizar símbolos de final de registro.

La representación en cadenas de bytes tal y como se ha discutido presenta varios inconvenientes:

- No resulta sencillo volver a utilizar el espacio ocupado anteriormente por un registro borrado. Aunque existen técnicas para gestionar la inserción y el borrado de registros, generan gran número de fragmentos pequeños de almacenamiento de disco desaprovechados.
- Por lo general no queda espacio para el aumento del tamaño de los registros. Si un registro de longitud variable aumenta de tamaño hay que desplazarlo (el movimiento resulta costoso si el registro está almacenado en otro lugar de la base de datos; por ejemplo, en los índices o en otros registros), ya que los punteros se deben localizar y actualizar.

Por tanto, no se suele utilizar la representación sencilla en cadenas de bytes tal y como aquí se ha descrito para implementar registros de longitud variable. Sin embargo, una forma modificada de la representación en cadenas de bytes, denominada estructura de páginas con ranuras, se utiliza normalmente para organizar los registros *dentro de* cada bloque.

La **estructura de páginas con ranuras** se muestra en la Figura 11.11. 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 *de manera contigua* en el bloque, empezando desde el final del mismo. 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.

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

FIGURA 11.10. Representación en cadenas de bytes de los registros de longitud variable.

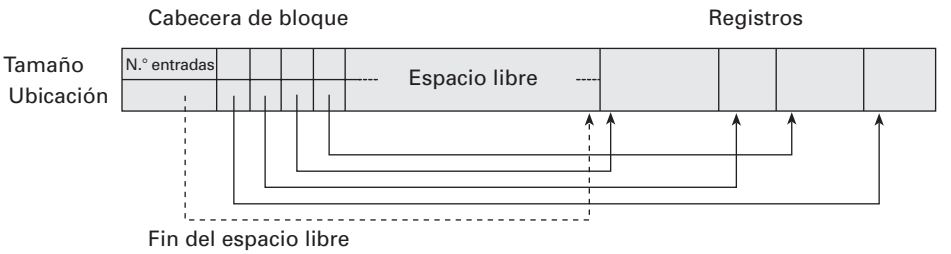


FIGURA 11.11. Estructura de páginas con ranuras.

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 de 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 utilizando técnicas parecidas, siempre y cuando quede espacio en el bloque. El coste de trasladar los registros no es demasiado elevado, dado que el tamaño del bloque es limitado: un valor típico es cuatro kilobytes.

La estructura de páginas con ranuras necesita que no haya punteros que apunten 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 indirección permite a los registros desplazarse para evitar la fragmentación del espacio dentro del bloque al tiempo que permite los punteros indirectos a los registros.

11.6.2.2. Representación de longitud fija

Otra manera de implementar eficientemente los registros de longitud variable en un sistema de archivos es utilizar uno o varios registros de longitud fija para representar cada registro de longitud variable.

Hay dos técnicas para hacer esto:

1. **Espacio reservado.** Si hay una longitud de registro máxima que no se supera nunca, se pueden utilizar registros de longitud fija de esa longitud. El espacio no utilizado (por los registros más cortos que el espacio máximo) se rellena con un símbolo especial de valor nulo o de final de registro.
2. **Representación con listas.** El registro de longitud variable se representa mediante una lista de registros de longitud fija, enlazada mediante punteros.

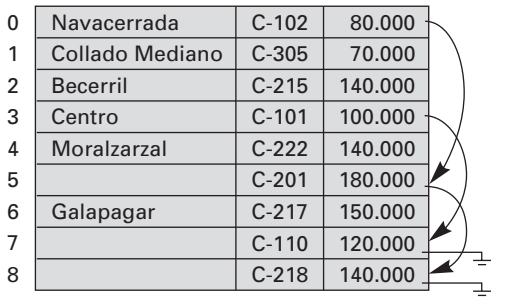
Si se escoge aplicar el método del espacio reservado al ejemplo de las cuentas bancarias hay que seleccionar una longitud de registro máxima. En la Figura 11.12 se muestra el modo en que se representaría el archivo si se permitiera un máximo de tres cuentas por sucursal. Los registros de este archivo son del tipo *lista-cuentas*, pero el *array* contiene exactamente tres elementos. Las sucursales con menos de tres cuentas (por ejemplo, Collado Mediano) tienen registros con campos con valores nulos. En la Figura 11.12 se utiliza el símbolo ⊥ para representar esta situación. En la práctica se utiliza un valor concreto que no pueda representar nunca un dato real (por ejemplo, un «número de cuenta» negativo o un nombre que comience por un «*»).

El método del espacio reservado resulta útil cuando la mayor parte de los registros son de una longitud cercana a la máxima. En caso contrario se puede desperdiciar una cantidad de espacio significativa. En el ejemplo bancario puede que algunas sucursales tengan muchas más cuentas que otras. Esta situación lleva a considerar el uso del método de las listas enlazadas. Para representar el archivo utilizando el método de los punteros se añade un campo puntero igual que se hizo en la Figura 11.9. La estructura resultante se muestra en la Figura 11.13.

Las estructuras de archivo de las Figuras 11.9 y 11.13 son idénticas, salvo que en la Figura 11.9 sólo se utilizaron los punteros para enlazar los registros borrados, mientras que en la Figura 11.13 se enlazan todos los registros pertenecientes a la misma sucursal.

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

FIGURA 11.12. El archivo de la Figura 11.10 utilizando el método del espacio reservado.



0	Navacerrada	C-102	80.000
1	Collado Mediano	C-305	70.000
2	Becerril	C-215	140.000
3	Centro	C-101	100.000
4	Moralzarzal	C-222	140.000
5		C-201	180.000
6	Galapagar	C-217	150.000
7		C-110	120.000
8		C-218	140.000

FIGURA 11.13. El archivo de la Figura 11.10 utilizando el método de las listas enlazadas.

Un inconveniente de la estructura de la Figura 11.13 es que se desperdicia espacio en todos los registros excepto en el primero de la serie. El primer registro debe tener el valor *nombre-sucursal*, pero los registros siguientes no necesitan tenerlo. No obstante, hay que incluir en todos los registros un campo para *nombre-sucursal*, o los registros no serán de longitud constante. El espacio desperdiciado es significativo, dado que se espera en la práctica que cada sucursal tenga un gran número de cuentas. Para resolver este problema se permiten en el archivo dos tipos de bloques:

1. **Bloque ancla**, que contiene el primer registro de cada cadena.
2. **Bloque de desbordamiento**, que contiene los registros que no son los primeros de sus cadenas.

Por tanto, todos los registros *del interior de cada bloque* tienen la misma longitud, aunque no todos los registros del archivo tengan la misma longitud. En la Figura 11.14 se muestra esta estructura de archivos.

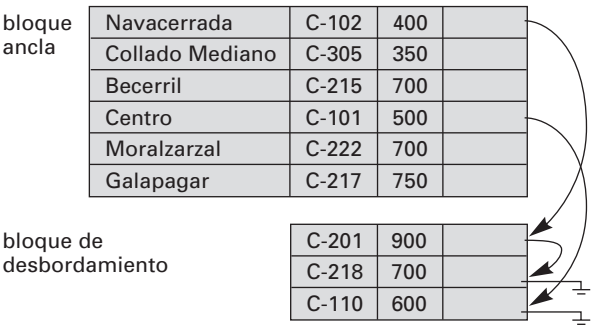


FIGURA 11.14. Estructuras de bloque ancla y de bloque de desbordamiento.

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 los archivos. Un conjunto de registros constituye un ejemplo de esta relación. Dado un conjunto de registros, la pregunta siguiente es la manera de organizarlos en archivos. A continuación se indican varias de las maneras de organizar los registros en archivos:

- **Organización de archivos en montículo.** En esta organización se puede colocar cualquier registro en cualquier parte del archivo en que haya espacio suficiente. No hay ninguna ordenación de los registros. Generalmente sólo hay un archivo por cada relación.
- **Organización de archivos secuenciales.** En esta organización los registros se guardan en orden secuencial, basado en el valor de la clave de búsqueda de cada registro. La implementación de esta organización se describe en el Apartado 11.7.1.
- **Organización asociativa (*hash*) de archivos.** En esta organización se calcula una función de asociación (*hash*) de algún atributo de cada registro. El resultado de la función de asociación especifica el bloque del archivo en que se deberá colocar el registro. Esta organización se describe en el Capítulo 12; está estrechamente relacionada con las estructuras para la creación de índices descritas en dicho capítulo.

Generalmente se usa un archivo separado para almacenar los registros de cada relación. Sin embargo, en una **organización de archivos en agrupaciones** 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 las dos relaciones se pueden considerar relacionados si casan en una reunión de las dos relaciones. Esta organización se describe en el Apartado 11.7.2.

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 una clave de búsqueda. Una **clave de búsqueda** es cualquier atributo o conjunto de atributos; no tiene por qué ser una clave primaria, ni siquiera una superclave. Para permitir la recuperación rápida de los registros según el orden de la clave de búsqueda, los registros se vinculan mediante punteros. El puntero de cada registro apunta 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 de acuerdo con el orden indicado

por la clave de búsqueda, o en un orden tan cercano a éste como sea posible.

En la Figura 11.15 se 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, utilizando como tal *nombre-sucursal*.

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

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

1. Localizar el registro del archivo que precede al registro que se va a insertar en el orden de la clave de búsqueda.
2. Si existe algún registro vacío (es decir, un espacio que haya quedado libre después de un 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.

En la Figura 11.16 se muestra el archivo de la Figura 11.15 después de la inserción del registro (C-888, Leganés, 800). La estructura de la Figura 11.16 permite la inserción rápida de nuevos registros, pero obliga a las aplicaciones de procesamiento de archivos secuenciales a procesar los registros en un orden que no coincide con su orden físico.

Si hay que guardar un número relativamente pequeño de registros en los bloques de desbordamiento, este enfoque funciona bien. Finalmente, sin embargo, la correspondencia entre el orden de la clave de búsqueda y el orden físico puede perderse totalmente, en cuyo

FIGURA 11.15. Archivo secuencial para los registros de *cuenta*.

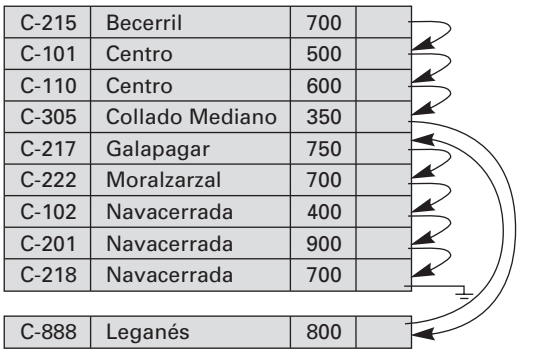


FIGURA 11.16. El archivo secuencial después de una inserción.

caso el procesamiento secuencial será significativamente 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 se necesitan las reorganizaciones depende de la frecuencia de inserción de registros nuevos. En el caso extremo en que las inserciones tengan lugar raramente, siempre resultará posible mantener el archivo en el orden físico correcto. En ese caso no es necesario el campo puntero mostrado en la Figura 11.15.

11.7.2. Organización de archivos en agrupaciones

Muchos sistemas de bases de datos relacionales guardan cada relación en un archivo diferente de modo que puedan aprovechar completamente el sistema de archivos que forma parte del sistema operativo. Generalmente las tuplas de cada relación pueden representarse como registros de longitud fija. Por tanto, las relaciones pueden hacerse corresponder con una estructura de archivos sencilla. Esta implementación sencilla de los sistemas de bases de datos relacionales resulta adecuada para los sistemas de bases de datos diseñados para computadoras personales. En estos sistemas el tamaño de la base de datos es pequeño, por lo que se obtiene poco provecho de una estructura de archivos avanzada. Además, en algunas computadoras personales el pequeño tamaño global del código objeto del sistema de bases de datos resulta fundamental. Una estructura de archivos sencilla reduce la cantidad de código necesaria para implementar el sistema.

Este enfoque sencillo de la implementación de 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 ventajas en el rendimiento mediante la asignación esmerada de los registros a los bloques y de la organización cuidadosa de los propios bloques. Por tanto, resulta evidente que puede resultar beneficiosa una estructura de archivos más compleja, aunque se mantenga la estrategia de guardar cada relación en un archivo diferente.

nombre-cliente	*número-cuenta*
López	C-102
López	C-220
López	C-503
Abril	C-305

FIGURA 11.17. La relación *impositor*.

Sin embargo, muchos sistemas de bases de datos de gran tamaño no utilizan directamente el sistema operativo subyacente para la gestión de archivos. Por el contrario, se asigna al sistema de bases de datos un archivo de gran tamaño del sistema operativo. En este archivo se guardan todas las relaciones y se confía la gestión de este archivo al sistema de bases de datos. Para ver 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 *cliente* con el mismo valor de *nombre-cliente*. En teoría, estos registros se podrán encontrar con la ayuda de los *índices*, que se discutirán en el Capítulo 12. Independientemente de la manera en que se encuentren estos 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.

Como ejemplo concreto, considérense las relaciones *impositor* y *cliente* de las Figuras 11.17 y 11.18, respectivamente. En la Figura 11.19 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

nombre-cliente	*calle-cliente*	*ciudad-cliente*
López	Principal	Arganzuela
Abril	Preciados	Valsaín

FIGURA 11.18. La relación *cliente*.

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

FIGURA 11.19. Estructura de archivo en agrupaciones.

tupla *cliente*, el bloque que contiene la tupla *cliente* también contiene las 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** es una organización de archivos, como la mostrada en la Figura 11.19 que almacena registros relacionados de dos o más relaciones en cada bloque. Esta organización permite leer muchos de los registros que satisfacen la condición de reunión utilizando un solo proceso de lectura de bloques. Por tanto, se puede procesar esta consulta concreta de manera más eficiente.

Este uso de la agrupación ha mejorado el procesamiento de una reunión particular (*impositor* $\bowtie$ *cliente*) pero ha producido el retardo del procesamiento de otros tipos de consulta. Por ejemplo,

select *
from *cliente*

necesita más accesos a los bloques que en el esquema en el que se guardaba cada relación en un archivo diferente. En lugar de que aparezcan varios registros de *cliente* en un mismo bloque, cada registro se halla en un bloque diferente. En realidad, hallar todos los registros de *cliente* no resulta posible sin alguna estructura adicional. Para encontrar todas las tuplas de la relación *cliente* en la estructura de la Figura 11.19 hay que vincular todos los registros de esa relación utilizando punteros, tal y como se muestra en la Figura 11.20.

La determinación del momento de utilizar la agrupación depende de los tipos de consulta que el diseñador de la base de datos considere más frecuentes. El uso cuidadoso de la agrupación puede producir ganancias de rendimiento significativas en el procesamiento de consultas.

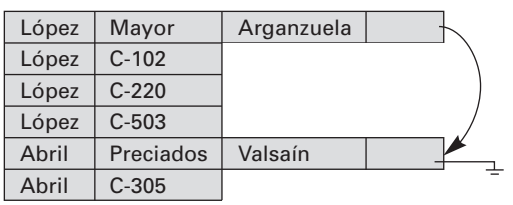


FIGURA 11.20. Estructura de archivo con agrupaciones con cadenas de punteros.

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 el esquema de las mismas. Esta información se denomina **diccionario de datos** o **catálogo del sistema**. Entre los tipos de información que debe guardar el sistema figuran los siguientes:

- Los nombres de las relaciones
- Los nombres de los atributos de cada relación
- Los dominios y las longitudes de los atributos
- Los nombres de las vistas definidas en la base de datos y las definiciones de esas vistas
- Las restricciones de integridad (por ejemplo, las restricciones de las claves)

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

- Los nombres de los usuarios autorizados
- La información de las cuentas de usuarios
- Contraseñas u otra información usada para autenticar a los usuarios

Además, se puede guardar información estadística y descriptiva sobre estos asuntos:

- Número de tuplas de cada relación
- Método de almacenamiento utilizado para cada relación (por ejemplo, con agrupaciones o sin agrupaciones)

El diccionario de datos puede también anotar la organización del almacenamiento (secuencial, asociativa o con montículos) de las relaciones y la ubicación donde se almacena cada relación:

- Si las relaciones se almacenan en archivos del sistema operativo, el diccionario no podría anotar los nombres de los archivos que almacenan cada relación.
- Si la base de datos almacena todas las relaciones en un único archivo, el diccionario puede anotar los bloques que almacenan los registros de cada relación en una estructura de datos como 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 el índice
- Los atributos sobre los que se define el índice
- 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 estructuras de datos y código 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 permite que se utilice toda la potencia de la base de datos en obtener un acceso rápido a los datos del sistema.

La elección exacta de la manera de representar los datos del sistema utilizando relaciones debe tomarla el diseñador del sistema. A continuación se ofrece una representación posible con las claves primarias subrayadas:

Metadatos-catálogo-sistema = (*nombre-relación*, *número-atributos*)
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*)

11.9. ALMACENAMIENTO PARA LAS BASES DE DATOS ORIENTADAS A OBJETOS**

Las técnicas de organización de los archivos descritas en el Apartado 11.7 (como las organizaciones en montículo, secuencial, asociativa y de agrupaciones) también pueden utilizarse para guardar los objetos de las bases de datos orientadas a objetos. Sin embargo, se necesitan características adicionales para poder trabajar con las propiedades de las bases de datos orientadas a objetos, como los campos de conjuntos y los punteros persistentes.

11.9.1. Correspondencia de los objetos con los archivos

La correspondencia de los objetos con los archivos tiene gran parecido con la correspondencia de las tuplas con los archivos de los sistemas relacionales. En el nivel inferior de la representación de los datos, tanto las partes de tuplas de los objetos como las de datos, son sencillamente secuencias de bytes. Por tanto, se pueden guardar los datos de los objetos utilizando las estructuras de archivos descritas en los apartados anteriores con algunas modificaciones que se indican a continuación.

Los objetos de las bases de datos orientadas a objetos pueden carecer de la uniformidad de las tuplas de las bases de datos relacionales. Por ejemplo, los campos de los registros pueden ser conjuntos, a diferencia de las bases de datos relacionales, en los que se suele exigir que los datos se encuentren (por lo menos) en la primera forma normal. Además, los objetos pueden ser muy grandes. Hay que tratar estos objetos de manera diferente de los registros de los sistemas relacionales.

Se pueden implementar campos de conjuntos que tengan un número pequeño de elementos que utilicen estructuras de datos como las listas enlazadas. Los campos de conjuntos que tienen un número de elementos mayor pueden implementarse como relaciones en la base de datos. Los campos de conjuntos también pueden borrarse en el nivel de almacenamiento mediante la normalización: se crea una relación que contenga una tupla para cada valor del campo de conjunto de un objeto. Cada tupla también contiene el identificador del objeto. El sistema de almacenamiento da a los niveles superiores del sistema de bases de datos el aspecto de un campo de conjuntos, aunque en realidad el campo de conjuntos se haya normalizado para crear una relación nueva.

Algunas aplicaciones incluyen objetos muy grandes que no se descomponen fácilmente en componentes menores. Cada uno de estos objetos de gran tamaño puede guardarse en un archivo diferente. Esta idea se discute en el Apartado 11.9.6.

11.9.2. Implementación de los identificadores de los objetos

Dado que los objetos se identifican mediante los identificadores de los objetos (IDO), los sistemas de almacenamiento de objetos necesitan un mecanismo para encontrar un objeto dado su IDO. Si los IDOs son **lógicos** (es decir, no especifican la ubicación del objeto) el sistema de almacenamiento debe tener un índice que asocie los IDOs con la ubicación real del objeto. Si los IDOs son **físicos** (es decir, codifican la ubicación del objeto) se puede encontrar el objeto directamente. Los IDOs físicos suelen tener las tres partes siguientes:

1. Un identificador de volumen o de archivo
2. Un identificador de las páginas dentro del volumen o archivo
3. Un desplazamiento dentro de la página

Además, los IDOs físicos pueden contener un **identificador único**, que es un entero que distingue el IDO de los identificadores de los demás objetos que se hayan guardado anteriormente en la misma ubicación y se borraron o se trasladaron a otra parte. El identificador único también se guarda con el objeto y deben coincidir los identificadores del IDO y los del objeto correspondiente. Si el identificador único de un IDO físico no coincide con el identificador único del objeto al que apunta el IDO, el sistema detecta que el puntero es un puntero colgante e indica un error (un **puntero colgante** es un puntero que no apunta a un objeto válido). En la Figura 11.21 se ilustra este esquema.

Estos errores de puntero tienen lugar cuando se utilizan de manera accidental IDOs físicos correspondientes a objetos antiguos que han sido borrados. Si el espacio ocupado por el objeto se ha vuelto a asignar, puede que haya un objeto nuevo en esa ubicación, y se puede dirigir a él de manera incorrecta el identificador del objeto antiguo. Si no se detecta el uso de los punteros colgantes se puede dar lugar al deterioro del objeto nuevo guardado en la misma ubicación. El identificador único ayuda a detectar estos errores, dado que los

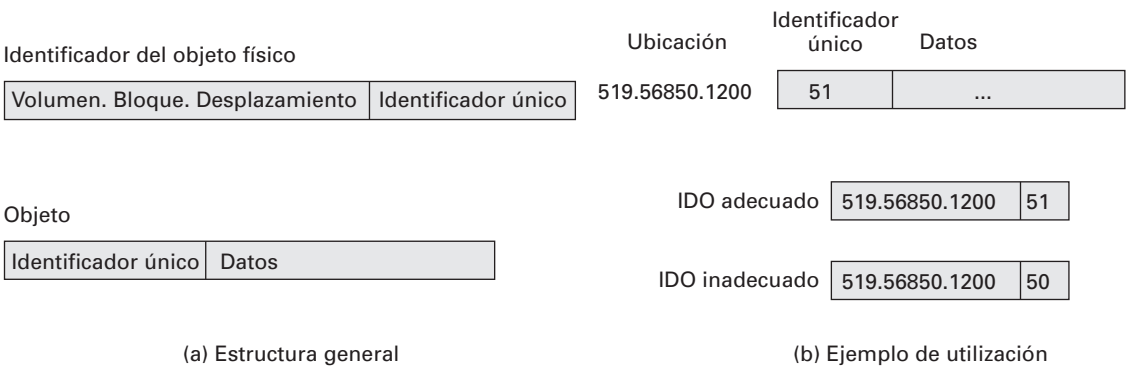


FIGURA 11.21. Identificadores únicos de un IDO.

identificadores únicos del IDO físico antiguo y el del nuevo objeto no coinciden.

Supóngase que hay que desplazar un objeto a una página nueva, quizás debido a que ha aumentado el tamaño del mismo y la página antigua no dispone de espacio adicional. En ese caso, el IDO físico apuntará a la página antigua, que ya no contiene el objeto. En vez de cambiar el IDO del objeto (lo que implica cambiar todos los objetos que apunten hacia él) se deja una **dirección de entrega** en la ubicación antigua. Cuando la base de datos intente encontrar el objeto encontrará la dirección de entrega en su lugar; entonces, utilizará la dirección de entrega para encontrar el objeto.

11.9.3. Gestión de los punteros persistentes

Los punteros persistentes se implementan en un lenguaje de programación persistente utilizando los IDOs. En algunas implementaciones los punteros persistentes son IDOs físicos; en otras, son IDOs lógicos. Una diferencia importante entre los punteros persistentes y los punteros internos de memoria es el tamaño de los mismos. Los punteros internos de memoria sólo necesitan tener el tamaño suficiente para apuntar a toda la memoria virtual. En las computadoras actuales los punteros internos de memoria tienen una longitud de cuatro bytes, que es suficiente para apuntar a cuatro gigabytes de memoria. Las nuevas arquitecturas tienen punteros de ocho bytes.

Los punteros persistentes tienen que apuntar a todos los datos de la base de datos. Dado que los sistemas de bases de datos suelen ser mayores que cuatro gigabytes, los punteros persistentes suelen tener una longitud de al menos ocho bytes. Muchas bases de datos orientadas a objetos proporcionan también identificadores únicos en los punteros persistentes para detectar las referencias colgantes. Esta característica aumenta aún más el tamaño de los punteros persistentes. Por tanto, los punteros persistentes son de una longitud considerablemente mayor que los punteros internos de memoria.

La acción de buscar un objeto dado su identificador se denomina **desreferenciar**. Dado un puntero interno de memoria (como en C++) buscar el objeto es simplemente una referencia a la memoria. Dado un puntero persistente, desreferenciar un objeto tiene una etapa adicional: hay que encontrar la ubicación real del objeto en la memoria buscando el puntero persistente en una tabla. Si el objeto no se halla todavía en la memoria hay que cargarlo desde el disco. Se puede implementar la búsqueda en la tabla de manera bastante eficiente utilizando funciones de asociación, pero la búsqueda sigue siendo lenta comparada con la desreferencia de los punteros, aunque el objeto ya se encuentre en memoria.

El **rescate de punteros** es una manera de reducir el coste de encontrar los objetos persistentes que ya se hallen en la memoria. La idea es que, cuando se desreferencia por primera vez un puntero persistente, se encuentra el objeto y se lleva a la memoria si es que no está ya allí. Ahora se da un paso adicional (se guarda un puntero interno de memoria para el objeto en lugar del puntero persistente). La siguiente ocasión en que se desreferencie el *mismo* puntero persistente la ubicación interna en la memoria puede leerse directamente, por lo que se evita el coste de encontrar el objeto. (En caso de que se puedan volver a desplazar los objetos persistentes de la memoria al disco para hacer sitio para otros objetos persistentes hay que dar otro paso más para asegurarse de que el objeto siga estando en la memoria). De manera análoga, cuando se copia al disco un objeto, hay que **devolver** todos los punteros persistentes que contenga y que se hubieran rescatado y hay que volver a convertirlos en su representación persistente. El rescate de los punteros al efectuar su desreferencia, tal y como se ha descrito aquí, se denomina **rescate software**.

La gestión de la memoria intermedia es más compleja si se utiliza el rescate de punteros, dado que la ubicación física de un objeto no debe cambiar una vez que se lleva ese objeto a la memoria intermedia. Una manera de asegurarse de que no cambiará es clavar las páginas que contengan objetos rescatados en la memoria intermedia, de manera que no sean sustituidas hasta que el programa que realizó el rescate haya concluido. Véanse las notas bibliográficas para tener información sobre esquemas de gestión de la memoria intermedia más complejos, que incluyen técnicas de correspondencia de la memoria virtual, que eliminan la necesidad de clavar las páginas de la memoria intermedia.

11.9.4. Rescate hardware

La existencia de dos tipos de punteros, persistentes y transitorios (internos de la memoria), resulta poco conveniente. Los programadores tienen que recordar el tipo de puntero y puede que tengan que escribir el código dos veces (una para los punteros persistentes y otra para los punteros internos de la memoria). Resultaría más conveniente que tanto los punteros persistentes como los internos de la memoria fueran del mismo tipo.

Una manera sencilla de combinar los tipos de puntero persistente e interno de la memoria es extender simplemente la longitud de los punteros internos de la memoria hasta el mismo tamaño que tienen los punteros persistentes y utilizar un bit del identificador para distinguir entre punteros persistentes e internos de la memoria. Sin embargo, el coste de almacenamiento de los punteros persistentes de mayor longitud tienen que soportarlo también los punteros internos de la memoria; por tanto, este esquema no se utiliza mucho.

Se describirá una técnica denominada **rescate hardware** que utiliza el hardware para la gestión de la memoria presente en la mayor parte de los sistemas informáticos actuales para abordar este problema. Cuando se accede a los datos de una página en memoria virtual y el sistema operativo detecta que la página no tiene almacenamiento real asignado, o que ha sido protegida contra accesos, entonces ocurre una **violación de la segmentación**[3]. Muchos sistemas operativos proporcionan

un mecanismo para especificar una función a la que se llama cuando sucede una violación de segmentación, y también un mecanismo para asignar almacenamiento para una página en el espacio virtual de direcciones y para establecer los permisos de acceso a la página. En la mayoría de sistemas Unix, la llamada al sistema mmap proporciona esta última característica. El rescate hardware hace un uso inteligente de los mecanismos anteriores.

El rescate hardware presenta dos ventajas fundamentales respecto al rescate software.

1. Puede guardar los punteros persistentes en los objetos en el mismo espacio que necesitan los punteros internos de la memoria (junto con el almacenamiento adicional externo al objeto).
2. Transforma de manera transparente los punteros persistentes en internos de la memoria, y viceversa, de forma inteligente y eficiente. El software escrito para trabajar con los punteros internos de la memoria puede, por tanto, trabajar también con los punteros persistentes sin que haya necesidad de efectuar ningún cambio.

11.9.4.1. Representación de punteros

El rescate hardware utiliza la siguiente representación de los punteros persistentes contenidos en los objetos que se hallan en el disco. Un puntero persistente se divide conceptualmente en dos partes: un identificador de página en la base de datos y un desplazamiento en la misma página 4. El identificador de la página es en realidad un puntero indirecto de pequeño tamaño: cada página (u otra unidad de almacenamiento) tiene una tabla de traducción que proporciona una asociación entre los identificadores de página cortos y los identificadores de página completos de la base de datos. El sistema tiene que buscar el identificador de página corto en los punteros persistentes de la tabla de traducción para encontrar el identificador de página completo.

La tabla de traducción, en el peor de los casos, sólo tendrá un tamaño igual que el número máximo de punteros que puedan contener los objetos de una página; con un tamaño de página de 4.096 y un tamaño de puntero de cuatro bytes el número máximo de punteros es 1.024. En la práctica, la tabla de traducción probablemente contenga muchos menos elementos que el número máximo (1.024 en este ejemplo) y no ocupe demasiado espacio. El identificador de página corto sólo tiene que tener los bytes necesarios para identificar una fila de la tabla; con un tamaño de tabla máximo de 1.024 sólo hacen falta diez bytes. Por tanto, basta con un número pequeño de bits para guardar el identificador de página corto. Por tanto, la tabla de traducción permite que los punteros persistentes que no sean cortos quepan en el mismo espacio que los punteros internos de la memoria. Aunque sólo hacen falta unos pocos bits para los identificadores de página cortos, se utilizan como tales todos los bits de los punteros internos de la memoria distintos de los de desplazamiento de página. Esta arquitectura, como se verá, facilita el rescate.

La representación del esquema de punteros persistentes se muestra en la Figura 11.22, en la que hay tres objetos en la página, cada uno de los cuales contiene un puntero persistente. La tabla de traducción muestra la asociación entre los identificadores de página cortos y los identificadores de página sin abreviar de la base de datos para cada uno de los identificadores de página cortos de estos punteros persistentes. Los identificadores de página de la base de datos se muestran en el formato *volumen.página.desplazamiento*.

Se guarda información adicional con cada página para que se puedan encontrar todos los punteros persistentes de la página. El sistema actualiza la información cuando se crea o borra algún objeto de la página. La necesidad de encontrar todos los punteros persistentes de cada página se pondrá de manifiesto más adelante.

11.9.4.2. Rescate de punteros en una página

Inicialmente no se ha iniciado ninguna página en memoria virtual. Las páginas de la memoria virtual se pueden asignar a las páginas de la base de datos antes de que realmente se carguen, como se verá enseguida. Las páginas de la base de datos se cargan en memoria virtual cuando el sistema de bases de datos necesite acceder a los datos de la página. Antes de que se cargue una página de la base de datos, el sistema asigna una página de memoria virtual para ella si no se hubiese asignado ya una. El sistema carga la página de la base de datos en la página de la memoria virtual que se ha asignado.

Cuando el sistema carga una página P de la base de datos en memoria virtual, se realiza el rescate de punteros en la página: se buscan todos los punteros persistentes contenidos en los objetos de la página P utilizando la información adicional guardada en la misma. Para cada puntero en la página se realizan las siguientes acciones (sea $\langle p_i, d_i \rangle$ el valor del puntero persistente, donde p_i es el identificador de página corto y d_i es el desplazamiento de la página, y sea P_i el identificador de página completo de p_i, hallado en la tabla de traducción de la página P).

[3] A veces se usa el término **fallo de página** en lugar de *violación de la segmentación*, aunque las violaciones de protección de acceso no se consideran como fallos de página.

[4] El término página se usa normalmente para referirse a la página de memoria real o virtual, y el término bloque se usa para referirse a los bloques de disco de la base de datos. En el rescate hardware deben ser del mismo tamaño y los bloques de la base de datos se buscan en las páginas de la memoria virtual. Se usarán los términos página y bloque para significar lo mismo.

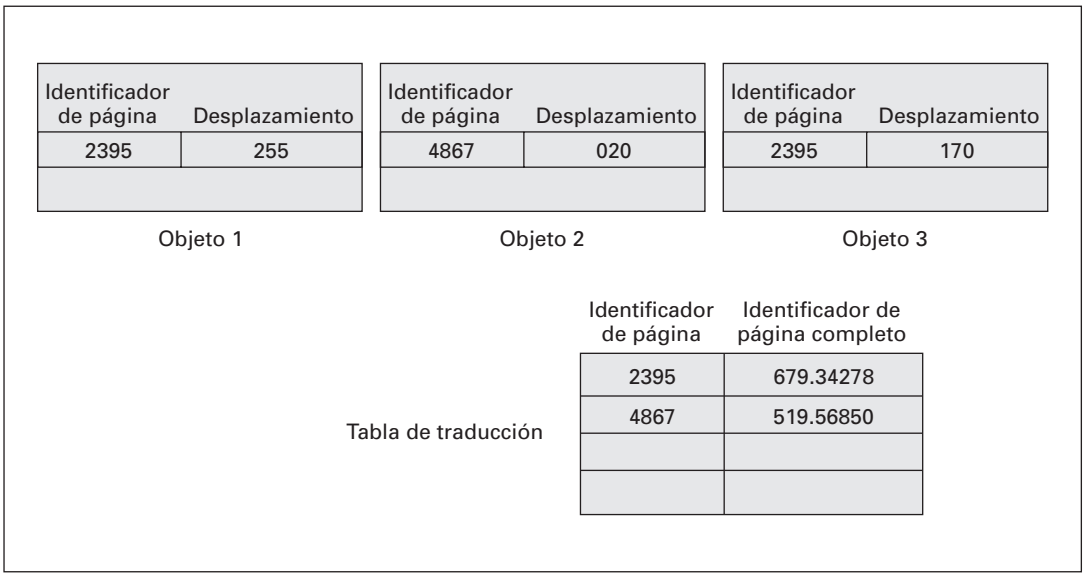


FIGURA 11.22. Imagen de una página antes del rescate.

1. Si la página P_i no tiene asignada todavía una página de memoria virtual se le asignará ahora una página libre del espacio de las direcciones virtuales. La página P_i residirá en la ubicación de la dirección virtual en el momento en que se lleve si esto se lleva a cabo. En este momento la página del espacio de direcciones virtuales no tiene espacio de almacenamiento asignado, ni en la memoria ni en el disco; se trata simplemente de un rango de direcciones reservado para la página de la base de datos. El sistema asigna espacio real cuando realmente carga la página P_i de la base de datos en memoria virtual.
2. Sea v_i la página de la memoria virtual asignada a P_i (bien con anterioridad, bien en el paso anterior). El sistema actualiza el puntero persistente considerado, cuyo valor es $\langle p_i, d_i \rangle$, reemplazando p_i por v_i.

Después de actualizar todos los punteros persistentes, cada entrada $\langle p_i, P_i \rangle$ de la tabla de traducción se reemplaza por $\langle v_i, P_i \rangle$, donde v_i es la página de memoria virtual que se ha asignado para P_i.

En la Figura 11.23 se muestra el estado de la página de la Figura 11.22 después de que se haya llevado a la memoria y se hayan rescatado sus punteros. Aquí se supone que la página cuyo identificador de página de la base de datos es 679.34278 se ha asociado con la página 5001 de la memoria, mientras que la página cuyo identificador es 519.56850 se ha hecho corresponder con la página 4867 (que coincide con el identificador de página corto). Todos los punteros de los objetos se han actua-

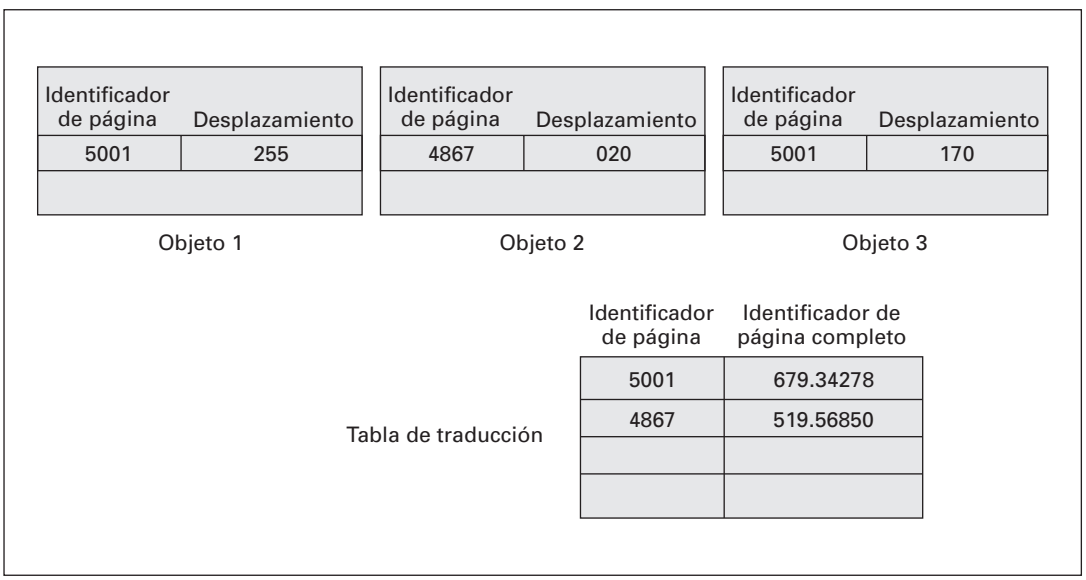


FIGURA 11.23. Imagen de la página después del rescate.

lizado para que reflejen la nueva asociación y pueden utilizarse como punteros internos de la memoria.

Al final de la fase de traducción de cada página, los objetos de la misma cumplen una propiedad importante: todos los punteros persistentes contenidos en los objetos de esa página se han transformado en punteros internos de la memoria. Por tanto, los objetos de las páginas internas de la memoria sólo contienen punteros internos de la memoria. Las rutinas que utilicen estos objetos ni siquiera tienen que conocer la existencia de los punteros persistentes. Por ejemplo, las bibliotecas existentes escritas para los objetos internos de la memoria pueden utilizarse sin modificación alguna para los objetos persistentes. Esto es una ventaja importante.

11.9.4.3. Desreferencia de punteros

Considérese la primera vez que se desreferencia un puntero interno de la memoria de una página *v*, cuando todavía no se ha asignado espacio de almacenamiento para esa página. Como se ha descrito, tendrá lugar una violación del encauzamiento y dará lugar a una llamada a una función en el sistema de bases de datos. El sistema de bases de datos realiza las siguientes acciones:

1. En primer lugar determina la página de la base de datos que se asignó a la página de la memoria virtual v_i; sea P_i el identificador de página sin acortar de la página de la base de datos (si no hay ninguna página de la base de datos asignada a v_i se indicará la existencia de un error).
2. Asigna espacio de almacenamiento para la página v_i y se carga en ella la página P_i de la base de datos.
3. Realiza rescate de punteros en la página P_i, el sistema permite que continúe la desreferencia del puntero que resultó en la violación de segmentación. La desreferencia del puntero encontrará cargado en memoria el objeto que estaba buscando.

Si cualquier puntero rescatado que apunte a un objeto en la página v_i se desreferencia más tarde, la desreferencia funciona igual que cualquier otro acceso a la memoria virtual, sin sobrecargas extra. En cambio, si no se usa el rescate, hay una considerable sobrecarga al localizar la página de la memoria intermedia que contiene el objeto y su posterior acceso. Esta sobrecarga aparece en *cada* acceso a los objetos de la página mientras que, cuando se usa el rescate, la sobrecarga sólo aparece en el *primer* acceso al objeto en la página. Los accesos posteriores funcionan a la velocidad normal de los accesos a memoria virtual. Por tanto, el rescate hardware proporciona excelentes beneficios de rendimiento para aplicaciones que desreferencien punteros repetidamente.

11.9.4.4. Optimizaciones

El rescate software tiene una operación de devolución asociada, cuando hay que volver a escribir en la base de datos una página de la memoria, para transformar de nuevo los punteros internos de la memoria en persistentes. El rescate hardware puede incluso evitar este paso (cuando se realiza el rescate de punteros de la página sencillamente se actualiza la tabla de traducción, por lo que la parte del identificador de página de los punteros internos de la memoria simulados puede utilizarse para buscar en la tabla). Por ejemplo, tal y como se muestra en la Figura 11.23, la página 679.34278 de la base de datos (con el identificador corto 2395 en la página mostrada) se asocia con la página 5001 de la memoria virtual. En este momento no sólo se actualiza el puntero del objeto 1 de 2395255 a 5001255, sino que también se actualiza a 5001 el identificador corto de la tabla. Por tanto, el identificador corto 5001 del objeto 1 y de la tabla vuelven a coincidir. Por consiguiente, se puede volver a escribir la página en el disco sin necesidad de devolverla.

Se pueden llevar a cabo varias optimizaciones del esquema básico aquí mostrado. Cuando se realiza el rescate de la página P, el sistema intenta asignar P' a la posición de dirección virtual indicada por el identificador de página corto de P' de la página P. Si la página puede asignarse tal y como se ha intentado, no hay que actualizar los punteros que la apunten. En el ejemplo de rescate expuesto, la página 519.56850 con el identificador de página corto 4867 se asoció con la página 4867 de la memoria virtual, que coincide con su identificador de página corto. Puede verse que no hay que modificar el puntero del objeto 2 que apunta a esta página durante el rescate. Si puede asignarse cada página a su posición correcta en el espacio de las direcciones virtuales no habrá que transformar ninguno de los punteros y se reducirá de modo significativo el coste del rescate.

El rescate hardware funciona aunque la base de datos sea de mayor tamaño que la memoria virtual, pero sólo mientras todas las páginas a las que cada proceso concreto tenga acceso quepan en la memoria virtual del mismo. Si no caben, habrá que sustituir las páginas llevadas a la memoria virtual, y esa sustitución resulta difícil, dado que puede haber punteros internos de la memoria que apunten a objetos de esas páginas. También puede utilizarse teóricamente el rescate hardware en el nivel de los conjuntos de páginas (a menudo denominados segmentos), en lugar de para páginas aisladas, siempre que los identificadores de página cortos, con los desplazamientos de página, no superen la memoria de los punteros internos de la memoria.

El rescate hardware también se puede usar en el nivel de los conjuntos de páginas (a menudo denominados segmentos) en lugar de en una sola página. Para el rescate por conjuntos el sistema usa una única tabla de traducción para todas las páginas del segmento. Carga las páginas en el segmento y las rescata cuando es necesario; no es necesario que se carguen todas juntas.

11.9.5. Estructura de los objetos en el disco o en la memoria

El formato con el que se guardan los objetos en la memoria puede ser diferente del formato con el que se guardan en el disco en la base de datos. Un motivo puede ser el uso del rescate software, en el que la estructura de los punteros persistentes y la de los internos de la memoria son diferentes. Otro motivo puede ser que se desee hacer que la base de datos sea accesible desde diferentes máquinas, posiblemente basadas en arquitecturas diferentes, y desde lenguajes diferentes, y desde programas compilados con compiladores diferentes, todo lo cual da lugar a diferencias en la representación en la memoria.

Considérese, por ejemplo, la definición de una estructura de datos en un lenguaje de programación como C++. La estructura física (como los tamaños y la representación de los enteros) del objeto es dependiente de la máquina en la que se ejecuta el programa[5]. Además, la estructura física puede depender también del compilador que se utilice; en un lenguaje tan complejo como C++ son posibles diferentes opciones para la traducción de la descripción de nivel superior a la estructura física, y cada compilador puede tomar sus propias opciones.

La solución a este problema es hacer independiente de la máquina y del compilador la representación física de los objetos de la base de datos. Los objetos pueden pasarse de la representación en el disco a las formas necesarias para la máquina, lenguaje y compilador concretos cuando se llevan a la memoria. Esta conversión puede hacerse de manera transparente al mismo tiempo que se rescatan los punteros del objeto, de modo que el programador no tenga que preocuparse por ella.

El primer paso en la implementación de un esquema así es definir un lenguaje común para describir la estructura de los objetos (es decir, un lenguaje para la definición de datos). Se han realizado varias propuestas, una de las cuales es el lenguaje de definición de objetos (Object Definition Language, ODL) desarrollado por el grupo de gestión de bases de datos de objetos (Object Database Management Group, ODMG). ODL tiene definidas asociaciones con Java, C++ y Smalltalk, por lo que en teoría los objetos de una base de datos que cumpla ODMG se pueden tratar utilizando cualquiera de estos lenguajes.

La definición de la estructura de cada clase de la base de datos se guarda (de manera lógica) en las bases de datos. El código para pasar los objetos de la base de datos a la representación de los mismos que trata el lenguaje de programación (y viceversa) es dependiente de la máquina y del compilador del lenguaje. Este código se puede generar de manera automática utilizando las definiciones de las clases de los objetos guardadas previamente.

Una fuente inesperada de diferencias entre las representaciones de los datos en el disco y en la memoria son los punteros ocultos de los objetos. Los **punteros ocultos** son punteros transitorios que generan los compiladores y se guardan en los objetos. Estos punteros apuntan (de modo indirecto) a las tablas utilizadas para implementar ciertos métodos del objeto. Las tablas suelen compilarse en código objeto ejecutable y la ubicación exacta de las tablas depende del código objeto ejecutable, por lo que puede ser diferente en procesos diferentes. Por tanto, cuando un proceso tiene acceso a un objeto, los punteros ocultos deben fijarse para que apunten a la ubicación correcta. Los punteros ocultos pueden inicializarse al tiempo que se llevan a cabo las conversiones entre las representaciones de los datos.

11.9.6. Objetos de gran tamaño

Los objetos pueden ser también enormemente grandes; por ejemplo, los objetos multimedia pueden ocupar varios megabytes. Los elementos de datos excepcionalmente grandes, como las secuencias de vídeo, pueden llegar a los gigabytes, aunque suelen dividirse en varios objetos, cada uno de ellos del orden de unos pocos megabytes o menos. Los objetos de gran tamaño que contienen datos binarios se denominan objetos de gran tamaño en binario (binary large objects, blobs), mientras que los grandes objetos que contienen datos de caracteres se denominan objetos de gran tamaño de tipo carácter (character large objects, clobs), como se vio en el Apartado 9.2.1.

La mayor parte de las bases de datos relacionales limitan el tamaño de los registros para que no tengan una longitud mayor que el tamaño de la página para simplificar la gestión de la memoria intermedia y del espacio libre. Los objetos de gran tamaño y los campos largos suelen guardarse en un archivo especial (o en un conjunto de archivos) reservado para el almacenamiento de campos largos.

Se presenta un problema en la gestión de los objetos de gran tamaño con la asignación de las páginas de

[5] Por ejemplo, las arquitecturas Motorola 680×0, la arquitectura IBM 360 y las arquitecturas Intel 80386/80486/Pentium/Pentium-II/Pentium-III tienen todas enteros de cuatro bytes. Sin embargo, se diferencian en la manera en que se disponen en las palabras los bits de los enteros. En las computadoras personales de las primeras generaciones los enteros tenían una longitud de dos bytes; en las arquitecturas de estaciones de trabajo más recientes, como la Alpha de Compaq, Itanium de Intel y UltraSparc de Sun, los enteros pueden tener una longitud de ocho bytes.

memoria intermedia. Los objetos de gran tamaño pueden necesitar ser guardados en una secuencia contigua de bytes cuando se llevan a la memoria; en este caso, si un objeto es mayor que una página, se deben asignar páginas contiguas de la memoria intermedia para guardarlo, lo que hace más difícil la gestión de la memoria intermedia.

Los objetos de gran tamaño suelen modificarse actualizando una parte de los mismos o insertándoles o borrándoles alguna parte del objeto, en lugar de escribiendo todo el objeto. Si hay que trabajar con inserciones o borrados, se pueden implementar los objetos de gran tamaño utilizando estructuras de árboles B (que se estudian en el Capítulo 12). Las estructuras de árboles B permiten leer todo el objeto, así como insertar y borrar partes del mismo.

Por razones prácticas se pueden manipular los objetos de gran tamaño utilizando programas de aplicaciones en vez de hacerlo dentro de la base de datos:

- **Datos de texto.** El texto suele tratarse como una cadena de bytes con la que trabajan los editores y los formateadores.
- **Datos gráficos.** Los datos gráficos pueden representarse como mapas de bits o como conjuntos de líneas, cuadros y otros objetos geométricos. Aunque algunos datos gráficos suelen tratarse dentro de la base de datos, en muchos casos se utiliza software de aplicaciones especiales, como en el diseño de circuitos integrados.
- **Datos de sonido y de vídeo.** Los datos de sonido y de vídeo suelen ser una representación digitalizada y comprimida creada y reproducida por software de aplicaciones diferentes. La modificación de los datos suele realizarse con software especial para ediciones, fuera del sistema de bases de datos.

El método más utilizado para actualizar estos datos es el método **desmarcar/marcar**. El usuario o la aplicación **desmarca** una copia de un objeto de campo largo, opera con esta copia utilizando programas especiales de aplicaciones y luego **marca** la copia modificada. Los conceptos *desmarcar* y *marcar* se corresponden a grandes rasgos con los de lectura y escritura. En algunos sistemas, al marcar se puede crear una nueva versión del objeto sin borrar la anterior.

11.10. RESUMEN

- En la mayor parte de los sistemas informáticos hay varios tipos de almacenamiento de datos. Estos medios de almacenamiento se clasifican según la velocidad con la que se puede tener acceso a los datos, el coste de adquisición de la memoria por unidad de datos y su fiabilidad. Entre los medios disponibles suelen estar la organización caché, la memoria principal, la memoria *flash*, los discos magnéticos, los discos ópticos y las cintas magnéticas.
- La fiabilidad de los medios de almacenamiento se determina mediante dos factores: si un corte en el suministro eléctrico o una caída del sistema hace que los datos se pierdan, y la probabilidad de fallo físico del dispositivo de almacenamiento.
- Se puede reducir la probabilidad del fallo físico conservando varias copias de los datos. Para los discos se puede utilizar la creación de imágenes. También se pueden usar métodos más sofisticados como las disposiciones redundantes de discos independientes (RAID). La distribución de los datos entre los discos ofrece altos índices de productividad en los accesos de gran tamaño; introduciendo la redundancia entre los discos se mejora mucho la fiabilidad. Se han propuesto varias organizaciones RAID diferentes, con características de coste, rendimiento y fiabilidad diferentes. Las organizaciones RAID de nivel 1 (la creación de imágenes) y RAID nivel 5 son las más utilizadas.
- Los *archivos* se pueden organizar lógicamente como una secuencia de registros asociados con bloques de disco. Un enfoque de la asociación de la base de datos con los archivos es utilizar varios archivos y guardar los registros de una única longitud fija en cualquier archivo dado. Una alternativa es estructurar los archivos de modo que puedan aceptar registros de longitud variable. Hay diferentes técnicas para la implementación de los registros de longitud variable, incluyendo el método de la página con ranuras, el método de los punteros y el método del espacio reservado.
- Dado que los datos se transfieren entre el almacenamiento en disco y la memoria principal en unidades de bloques, merece la pena asignar los registros de los archivos de modo que cada bloque contenga registros relacionados. Si se puede tener acceso a varios de los registros deseados utilizando sólo un acceso a bloques se evitan accesos al disco. Dado que los accesos al disco suelen ser el cuello de botella del rendimiento de los sistemas de bases de datos, la esmerada asignación de los registros a los bloques puede ofrecer mejoras significativas del rendimiento.
- Una manera de reducir el número de accesos al disco es guardar todos los bloques posibles en la memoria principal. Dado que no se pueden guardar todos los bloques en la memoria principal, hay que gestionar la asignación del espacio disponible en la memoria principal para el almacenamiento de los bloques. La

memoria intermedia (*buffer*) es la parte de la memoria disponible para el almacenamiento de las copias de los bloques del disco. El subsistema responsable de la asignación del espacio de la memoria intermedia se denomina *gestor de la memoria intermedia*.

- Los sistemas de almacenamiento para las bases de datos orientadas a objetos son algo diferentes de los sistemas de almacenamiento para las bases de datos relacionales: deben trabajar con objetos de gran tamaño, por ejemplo, y con punteros persistentes. Hay esquemas para detectar los punteros persistentes colgantes.
- Los esquemas de rescate software y hardware permiten la desreferencia eficiente de los punteros persistentes. Los esquemas basados en hardware utilizan el apoyo a la gestión de la memoria virtual realizado en hardware y muchos sistemas operativos de la generación actual los hacen accesibles a los programas del usuario.

TÉRMINOS DE REPASO

- Almacenamiento terciario
  - — Discos ópticos
  - — Cintas magnéticas
  - — Cambiadores automáticos
- Archivo
- Bloque de disco
- Catálogo del sistema
- Clave de búsqueda
- Diccionario de datos
- Disco magnético
  - — Plato
  - — Discos rígidos
  - — Disquetes
  - — Pistas
  - — Sectores
  - — Cabeza de lectura y escritura
  - — Brazo del disco
  - — Cilindro
  - — Controlador de discos
  - — Comprobación de suma
  - — Reasignación de sectores defectuosos
- Disposición redundante de discos independientes (RAID)
  - — Creación de imágenes
  - — Distribución de datos
  - — Distribución en el nivel de bit
  - — Distribución en el nivel de bloque
- Estructuras de almacenamiento para BDOO
- Identificador del objeto (IDO)
  - — IDO lógico
  - — IDO físico
  - — Identificador único
  - — Puntero colgante
  - — Dirección de entrega
- Intercambio en caliente
- Medidas de rendimiento de los discos
  - — Tiempo de acceso
  - — Tiempo de búsqueda
  - — Latencia rotacional
  - — Velocidad de transferencia de datos
  - — Tiempo medio entre fallos
- Medios de almacenamiento físico
  - — Caché
  - — Memoria principal
  - — Memoria flash
  - — Disco magnético
  - — Almacenamiento óptico
- Memoria intermedia (buffer)
  - — Gestor de la memoria intermedia
  - — Bloques clavados
  - — Salida forzada de bloques
- Niveles de RAID
  - — Nivel 0 (distribución de bloques sin redundancia)
  - — Nivel 1 (distribución de bloques con creación de imágenes)
  - — Nivel 3 (distribución de bits con paridad)
  - — Nivel 5 (distribución de bloques con paridad distribuida)
  - — Nivel 6 (distribución de bloques con redundancia P + Q)
- Objetos de gran tamaño
- Optimización del acceso a bloques de disco
  - — Planificación del brazo
  - — Algoritmo del ascensor
  - — Organización de archivos
  - — Desfragmentación
  - — Memorias intermedias de escritura no volátiles
  - — Memoria no volátil de acceso aleatorio
  - — Disco del registro histórico
  - — Sistema de archivos basado en registro histórico

- Organización asociativa (hash) de archivos
- Organización de archivos
  - — Lista libre
- Organización de archivos en agrupaciones
- Organización de archivos en montículo
- Organización de archivos secuenciales
- Políticas de sustitución de la memoria intermedia
  - — Menos recientemente utilizado (Least Recently Used, LRU)
  - — Extracción inmediata
  - — Más recientemente utilizado (Most Recently Used, MRU)
- Punteros ocultos
- RAID hardware
- RAID software
- Registros de longitud fija
  - — Representación en cadenas de bytes
  - — Estructura de páginas con ranuras
  - — Espacio reservado
  - — Representación de listas
- Registros de longitud variable
  - — Cabecera del archivo
  - — Lista libre
- Rendimiento de la reconstrucción
- Rescate de punteros
  - — Desreferencia
  - — Devolución
  - — Rescate software
  - — Rescate hardware
  - — Violación de la segmentación
  - — Fallo de página

EJERCICIOS

11.1. Indíquense los medios de almacenamiento físico disponibles en las computadoras que se utilizan habitualmente. Dese la velocidad con la que se puede tener acceso a los datos en cada medio.

11.2. ¿Cómo afecta la reasignación de los sectores dañados por los controladores de disco a la velocidad de recuperación de los datos?

11.3. Considérese la siguiente disposición de los bloques de datos y de paridad de cuatro discos:

Disco 1	Disco 2	Disco 3	Disco 4
B_1	B_2	B_3	B_4
P_1	B_5	B_6	B_7
B_8	P_2	B_9	B_{10}
⋮	⋮	⋮	⋮

B_i representa los bloques de datos; P_i, los bloques de paridad. El bloque de paridad P_i es el bloque de paridad para los bloques de datos B_{4i-3} a B_{4i}. Indíquense los problemas que puede presentar esta disposición.

11.4. Un fallo en el suministro eléctrico que se produzca mientras se escribe un bloque del disco puede dar lugar a que el bloque sólo se escriba parcialmente. Supóngase que se pueden detectar los bloques escritos parcialmente. Un proceso atómico de escritura de bloque es aquel en el que se escribe el bloque entero o no se escribe nada (es decir, no hay procesos de escritura parciales). Propónganse esquemas para conseguir el efecto de los procesos atómicos de escritura con los siguientes esquemas RAID. Los esquemas deben implicar procesos de recuperación de fallos.

a. RAID de nivel 1 (creación de imágenes)

b. RAID de nivel 5 (entrelazado de bloques, paridad distribuida)

11.5. Los sistemas RAID suelen permitir la sustitución de los discos averiados sin que se impida el acceso al sistema. Por tanto, los datos del disco averiado deben reconstruirse y escribirse en el disco de repuesto mientras el sistema se halla en funcionamiento. ¿Con cuál de los niveles RAID es menor la interferencia entre los accesos al disco reconstruido y los accesos al resto de los discos? Justifíquese la respuesta.

11.6. Dese un ejemplo de una expresión de álgebra relacional y de una estrategia de procesamiento de consultas en cada una de las situaciones siguientes:

a. MRU es preferible a LRU.

b. LRU es preferible a MRU.

11.7. Considérese el borrado del registro 5 del archivo de la Figura 11.8. Compárense las ventajas relativas de las siguientes técnicas para implementar el borrado:

a. Trasladar el registro 6 al espacio ocupado por el registro 5 y desplazar el registro 7 al espacio ocupado por el registro 6.

b. Trasladar el registro 7 al espacio ocupado por el registro 5.

c. Marcar el registro 5 como borrado y no desplazar ningún registro.

11.8. Muéstrese la estructura del archivo de la Figura 11.9 después de cada uno de los pasos siguientes:

a. Insertar (C-323, Galapagar, 1600).

b. Borrar el registro 2.

c. Insertar (C-626, Galapagar, 2000).

11.9. Dese un ejemplo de una aplicación de bases de datos en que sea preferible el método del espacio reservado para la representación de los registros de longitud variable frente al método de los punteros. Justifíquese la respuesta.

11.10. Dese un ejemplo de una aplicación de bases de datos en la que sea preferible el método de los punteros para representar los registros de longitud variable al método del espacio reservado. Justifíquese la respuesta.

11.11. Muéstrese la estructura del archivo de la Figura 11.12 después de cada uno de los pasos siguientes:

a. InsertFar (C-101, Becerril, 2800).
b. Insertar (C-323, Galapagar, 1600).
c. Borrar (C-102, Navacerrada, 400).

11.12. ¿Qué ocurre si se intenta insertar el registro

(C-929, Navacerrada, 3000)

en el archivo de la Figura 11.12?

11.13. Muéstrese la estructura del archivo de la Figura 11.13 después de cada uno de los pasos siguientes:

a. Insertar (C-101, Becerril, 560.000).
b. Insertar (C-323, Galapagar, 320.000).
c. Borrar (C-102, Navacerrada, 80.000).

11.14. Explíquese por qué la asignación de los registros a los bloques afecta de manera significativa al rendimiento de los sistemas de bases de datos.

11.15. Si es posible, determínese la estrategia de gestión de la memoria intermedia de su sistema operativo ejecutándose en su computadora y los mecanismos que proporciona para controlar la sustitución de páginas. Discútase cómo el control sobre la sustitución que proporciona podría ser útil para la implementación de sistemas de bases de datos.

11.16. En la organización secuencial de los archivos, ¿por qué se utiliza un bloque de desbordamiento aunque sólo haya en ese momento un único registro de desbordamiento?

11.17. Indíquense dos ventajas y dos inconvenientes de cada una de las estrategias siguientes para el almacenamiento de bases de datos relacionales:

a. Guardar cada relación en un archivo.
b. Guardar varias relaciones (quizá toda la base de datos) en un archivo.

11.18. Considérese una base de datos relacional con dos relaciones:

curso (*nombre-curso*, *aula*, *profesor*)
matrícula (*nombre-curso*, *nombre-estudiante*, *nivel*)

Defínanse ejemplos de estas relaciones para tres cursos, en cada uno de los cuales se matriculan cinco estudiantes. Dese una estructura de archivos de estas relaciones que utilice la agrupación.

11.19. Considérese la siguiente técnica de mapa de bits para realizar el seguimiento del espacio libre de un archivo. Por cada bloque del archivo se mantienen dos bits en el mapa. Si el bloque está lleno entre el 0 y el 30 por ciento, los bits son 00, entre 30 por ciento y 60 por ciento, 01, entre 60 por ciento y 90 por ciento, 10, y por encima de 90 por ciento, 11. Tales mapas se pueden mantener en memoria incluso para grandes archivos.

a. Descríbase cómo mantener actualizado el mapa de bits al insertar y eliminar registros.
b. Descríbanse el beneficio de la técnica de los mapas de bits sobre las listas libres al buscar espacio libre y al actualizar su información.

11.20. Dese una versión normalizada de la relación *Metadatos-índices* y explíquese por qué al usar la versión normalizada se incurriría en pérdida de rendimiento.

11.21. Explíquese el motivo de que un IDO físico deba contener más información que un puntero que apunte a una ubicación física de almacenamiento.

11.22. Si se utilizan IDOs físicos, un objeto se puede reubicar guardando un puntero a su nueva ubicación. En el caso de que se guarden varios punteros para un objeto, ¿cuál sería el efecto sobre la velocidad de recuperación?

11.23. Defínase el término *puntero colgante*. Descríbase la manera en que el esquema de identificador único ayuda a detectar los punteros colgantes en las bases de datos orientadas a objetos.

11.24. Considérese el ejemplo de la página 276, que muestra que no hace falta el rescate si se utiliza el rescate hardware. Explíquese el motivo de que, en ese ejemplo, resulte seguro cambiar el identificador corto de la página 679.34278 de 2395 a 5001. ¿Puede tener ya alguna otra página el identificador corto 5001? Si fuera así, ¿cómo se resolvería esa situación?

NOTAS BIBLIOGRÁFICAS

Patterson y Hennessy [1995] discuten los aspectos de hardware de la memoria intermedia con traducción anticipada, de las cachés y de las unidades de gestión de la memoria. Rosch y Wethington [1999] presentan una excelente visión general del hardware de computadoras, incluyendo un tratamiento extensivo de todos los tipos de tecnologías de almacenamiento como disquetes, discos magnéticos, discos ópticos, cintas e interfaces de almacenamiento. Ruemmler y Wilkes [1994] presentan una revisión de lo tecnología de los discos magnéticos. La memoria flash se discute en Dippert y Levy [1993].

Las especificaciones de las unidades de disco actuales se pueden obtener de los sitios Web de sus fabricantes, como IBM, Seagate y Maxtor.

Las organizaciones alternativas de los discos que proporcionan un elevado grado de tolerancia a los fallos incluyen las desarrolladas por Gray et al. [1990] y por Bitton y Gray [1988]. La distribución en los discos se describe en Salem y García-Molina [1986]. Se presentan discusiones sobre las disposiciones redundantes de discos independientes (RAID) en Patterson et al. [1988] y en Chen y Patterson [1990]. Chen et al. [1994] pre-

sentan una excelente revisión de los principios y de la aplicación de RAID. Los códigos de Reed-Solomon se tratan en Pless [1989]. El sistema de archivos basado en registro histórico, que hace secuencial el acceso a disco, se describe en Rosenblum y Ousterhout [1991].

En los sistemas que permiten la informática portátil se pueden transmitir los datos de manera reiterada. El medio de transmisión puede considerarse un nivel de la jerarquía de almacenamiento (como un disco transmisor con latencia elevada). Estos aspectos se discuten en Acharya et al. [1995]. La gestión de la caché y de las memorias intermedias en la informática portátil se discute en Barbará e Imielinski [1994]. En Douglis et al. [1994] aparecen más discusiones de los problemas de almacenamiento en la informática portátil.

Las estructuras de datos básicas se discuten en Cormen et al.[1990]. Hay varios artículos que describen la estructura de almacenamiento de sistemas específicos de bases de datos. Astrahan et al. [1976] y System R. Chamberlin et al. [1981] repasan en retrospectiva, System R. *Oracle 8 Concepts Manual* (Oracle [1997]) describe la organización de almacenamiento del sistema de bases de datos Oracle 8. La estructura de Wisconsin Storage System (WiSS) se describe en Chou et al. [1985]. En Finkelstein et al. [1988] se describe una herramienta de software para el diseño físico de bases de datos relacionales.

La gestión de las memorias intermedias se discute en la mayor parte de los textos sobre sistemas operativos, incluido Silberschatz y Galvin [1994]. Stonebraker [1981] discute la relación entre los gestores de memoria intermedia de los sistemas de bases de datos y los de los sistemas operativos. Chou y DeWitt [1985] presentan algoritmos para la gestión de memoria intermedia en sistemas de bases de datos y describe un método de medida del rendimiento. Bridge et al. [1997] describen técnicas usadas en el gestor de la memoria intermedia del sistema de bases de datos Oracle.

En Wilson [1990], Moss [1990] y White y DeWitt [1992] se ofrecen descripciones y comparaciones del rendimiento de diferentes técnicas de rescate. White y DeWitt [1994] describen el esquema de gestión de memoria intermedia asociado con la memoria virtual utilizado en el sistema ObjectStore OODB y en el gestor de almacenamiento QuickStore. Utilizando este esquema se pueden asociar las páginas del disco con direcciones fijas de la memoria virtual, aunque no estén clavadas en la memoria intermedia. El gestor de almacenamiento de objetos Exodus se describe en Carey et al. [1986]. Biliris y Orenstein [1994] proporcionan una revisión de los sistemas de almacenamiento para bases de datos orientadas a objetos. Jagadish et al. [1994] describen un gestor de almacenamiento para bases de datos en memoria principal.