gonzz_
www.wuolah.com/student/gonzz_

tema4completolibro.pdf

Tema-4 completo del libro

2° Fundamentos de Bases de Datos

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

TEMA - 4: NIVEL INTERNO

3.1. INTRODUCCIÓN

Nivel interno: expresa en última instancia, las operaciones sobre los datos (creación, alteración y recuperación) en términos de actuación sobre unidades mínimas de almacenamiento denominadas páginas de base de datos. Este nivel provee al administrador mecanismos para optimizar el almacenamiento y el acceso a los datos. Está implementado en el SGBD.

Nivel físico: implementado en el sistema operativo, proporciona al SGBD una capa de abstracción sobre el hardware. El nivel interno del SGBD realiza el acceso a los medios de almacenamiento mediante llamadas a los servicios del sistema de archivos proporcionado por el SO.

En este tema se considerarán los mecanismos que pueden intervenir en la eficiencia de un SGBD sobre los que puede tomar algunas decisiones el administrador en relación con los aspectos de almacenamiento y de acceso a datos.

3.2. DISPOSITIVOS DE ALMACENAMIENTO

A la hora de organizar un sistema informático podemos encontrar una gran diversidad de tipos de almacenamiento, que se pueden agrupar según características comunes y criterios como *velocidad* (tiempo de acceso a los datos) *coste* ($/Mb), *capacidad*, y *fiabilidad* del dispositivos (tiempo medio entre fallos).

El almacenamiento se suele clasificar como primario (memoria principal y caché), que son más caras y son volátiles; secundario, que engloba a dispositivos como discos duros magnéticos, unas 1000 veces más lentos que los otros pero con más capacidad y no volátiles, además de permitir el acceso directo; y el almacenamiento terciario, como discos ópticos y cintas magnéticas, que tiene un tiempo de acceso muy alto y se puede usar para realizar copias de seguridad de BDs de mediana capacidad.

3.2.1. Estructura lógica de los discos duros

El nivel interno organiza el almacenamiento y el acceso a la BD usando discos duros y atendiendo a su estructura. Las unidades de disco duro están constituidas por un conjunto de discos magnéticos insertados en un eje de rotación solidario. Sobre cada una de las caras de los discos hay un cabezal de lecto/escritura dotado de movimiento transversal. Cuando la unidad está en funcionamiento, el eje del disco y por tanto cada una de las caras del mismo gira a una velocidad constante (de entre 4200 y 15000 rpm dependiendo del modelo). La estructura física del disco determina cómo se accede a la información. Cada cara está compuesta por un conjunto de pistas concéntricas que a su vez están divididas en sectores, que representan la unidad mínima de almacenamiento (o bloque). Todos los sectores de una unidad tienen la misma capacidad de almacenamiento independientemente de su ubicación en el disco. De esta forma, para acceder a un sector concreto de la unidad es preciso indicar en qué cilindro se encuentras, el número de sector en dicho cilindro y cuál de las superficies de los discos lo alberga.

Las unidades de disco, por lo general, suelen tener una caché intermedia que emplean para optimizar las transferencias de datos entre el disco y los buses de E/S de los ordenadores. Los principales parámetros que cualifican a una unidad de disco duro son su capacidad de almacenamiento; el tiempo medio de acceso (que mide en ms el tiempo medio transcurrido entre una instrucción y la obtención de los datos); la velocidad sostenida de escritura y lectura (en MB/s) y la velocidad de rotación, medida en rpm; además puede interesar considrar el tiempo medio entre fallos.

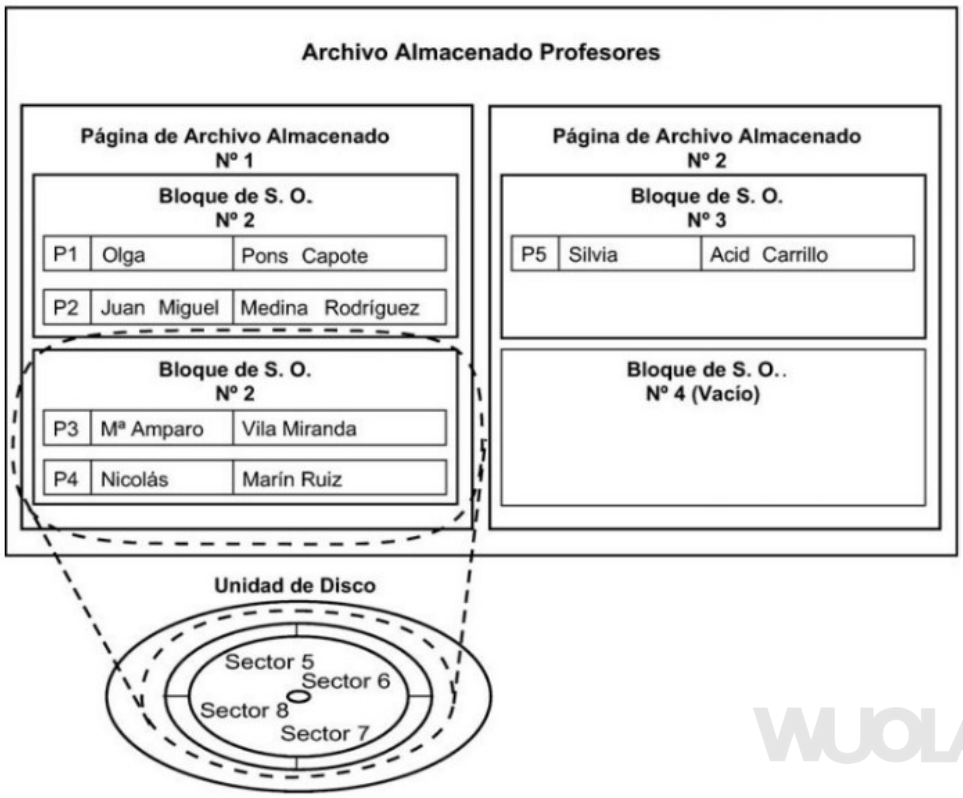
Las características fundamentales de los discos duros que se deben tener presentes cuando se implementa el acceso a los datos en el nivel interno son la posibilidad de acceso directo y que el sector representa la unidad de capacidad mínima para una transferencia a/desde el disco duro.

3.2.2. La memoria principal

La memoria principal es el dispositivo de almacenamiento primario de los computadores. Es volátil, y en esta se ubican los datos que precisan los programas para su ejecución. En un SGBD se puede usar para realizar una caché de la porción de la base de datos de uso más reciente para acelerar el procesamiento, almacenar los parámetros de estado o almacenar el código ejecutable de las sentencias de datos usadas. Desde el punto de vista del nivel interno se considera como un almacenamiento intermedio que ubica temporalmente los datos afectados por las operaciones del SGBD.

3.3. MÉTODOS DE ACCESO A LA BASE DE DATOS

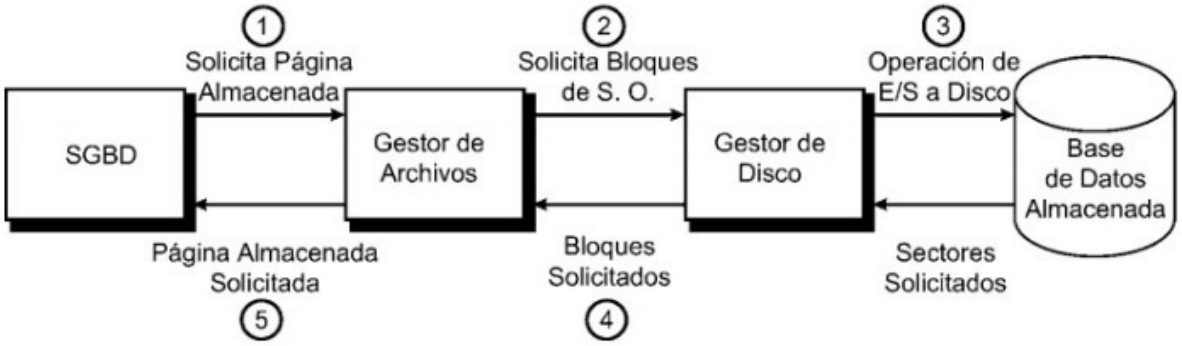
Según la descripción de la arquitectura de tres niveles los datos a nivel interno se consideran en términos de archivo, registro y campo almacenado. El carácter dinámico de la información contenida en un archivo almacenado precisa que éste se organice en fragmentos llamados páginas de archivo almacenado. Cada página puede contener, dependiendo del tamaño, desde una fracción de registro almacenado a varios registros. A nivel de sistema operativo cada una de estas páginas está integrada por varios fragmentos llamados bloques de SO. Por último, un bloque está compuesto por un número determinado de sectores de disco.



El *método de acceso* a la BD describe de qué manera se transforma un registro almacenado en una representación física a nivel de almacenamiento secundario. La implementación de esta transformación puede diferir de un SGBD a otro.

El acceso a los registros almacenados puede realizarse de dos formas:

- **Secuencial:** en el que, como consecuencia de una operación a nivel conceptual o externo, el SGBD determina que debe recuperar *todos* los registros de un archivo almacenado. Para esto el SGBD dispone de una página de direcciones en la que se relacionan los archivos almacenados disponibles junto con la dirección de la página de comienzo de cada uno.
- **Directa:** el procesamiento de una operación realizada a nivel conceptual o externo produce como resultado un identificador que permite localizar y recuperar de forma directa un registro almacenado concreto. Este proceso se desglosa en los siguientes pasos:
    1. El SGBD determina que debe recuperar un registro concreto del almacenamiento secundario. Da al gestor de archivos la dirección de la página que lo contiene.
    2. El gestor de archivos del SGBD, que tiene constancia de qué bloques de SO integran la página en cuestión, los solicita al gestor de disco del SO.
    3. El gestor de disco determina los sectores que integran las páginas mencionadas y las solicita a la unidad de disco.
    4. Una vez que obtiene esos sectores los agrupa en los bloques apropiados y los devuelve al gestor de archivos.
    5. Este, de la página recuperada, extrae y devuelve al SGBD el registro almacenado solicitado.



Para que el gestor de archivos del SGBD pueda localizar en qué página se encuentra el registro almacenado y el desplazamiento dentro de la página se usa la información que registra el RID (Record IDentifier), que almacena el número de página en que se encuentra el registro almacenado y un puntero a una posición, dentro de un vector de direcciones de la página, que contiene la dirección de comienzo del registro almacenado.

Un registro almacenado tiene una estructura compuesta por una *cabecera*, en la que se describen el número y el tipo de los campos almacenados que lo integran junto con su *contenido*. Las páginas de la BD deben tener un tamaño múltiplo del tamaño de bloque del sistema operativo.

Como una página determina al menos una operación de E/S, para optimizar el rendimiento de las operaciones sobre la BD, debemos organizar la estructura de almacenamiento y acceso de forma que las operaciones involucren la mínima cantidad posible de páginas de la BD, y por tanto de operaciones de E/S.

3.3.1. El gestor de disco del SO

En la mayoría de implementaciones, el SGBD interactúa con la BD almacenada en el almacenamiento secundario a través de los servicios proporcionados por el gestor de disco. Este se encarga de organizar la información que alberga la unidad de disco en términos de conjuntos de bloques, en los que cada conjunto representa un archivo a nivel de SO. Una BD puede tener uno o varios de estos archivos para almacenar su contenido. El gestor de disco también gestiona el espacio libre manteniendo conjunto de bloques libres. Las funciones más elementales que proporciona son:

- Crear un nuevo archivo de sistema operativo, es decir de un conjunto de bloques libres obtener un bloque cabecera y registrar el nombre del archivo la ubicación de este en el directorio principal del disco.
- Devolver el bloque *b* del conjunto de bloques *c*.
- Reemplazar el bloque *b* dentro del conjunto de bloques *c* por otro diferente.
- Añadir un nuevo bloque al conjunto de bloques *c*. Implica incrementar el tamaño de *c* y por tanto del archivo asociado usando un bloque libre.
- Eliminar el bloque *b* del conjunto de bloques *c*. Conlleva desvincular dicho bloque del conjunto *c* y asociarlo al conjunto de bloques libres.
- Eliminar un archivo de sistema operativo existente. Se localiza el bloque cabecera y se asocian al conjunto de bloques libres este bloque cabecera junto con todos los bloques que integran el archivo a eliminar. Por último se elimina del directorio las referencias al archivo.

3.3.2. El gestor de archivos del SGBD

Este se encarga de trasladar las representaciones en términos de campos, registros y archivos almacenados a representaciones en términos de bloques y conjuntos de bloques comprensibles por el gestor de disco del SO. Un requisito básico es la organización de los datos de manera que se minimice el tiempo de recuperación de los mismos bajo diferentes criterios. Esto hace que los SGBD necesiten sus propios gestores de archivos optimizados para el acceso flexible y eficiente a los datos. Sus funciones básicas son:

- Crear un nuevo archivo almacenado *a*. Esto conlleva asociar al archivo un conjunto de páginas de la BD compuesto al menos por una cabecera que identifique al archivo almacenado.
- Eliminar el archivo almacenado *a*.

- Recuperar el registro almacenado *r* del archivo almacenado *a*. Normalmente el SGBD proporciona el RID al invocar esta operación, por lo que el gestor de archivos solo tiene que obtener la página que contiene el registro para extraerlo.
- Añadir un nuevo registro almacenado al archivo almacenado *a*. Para ubicarlo, el gestor de archivos localiza la página de BD más apropiada entre las que pertenecen al archivo almacenado. Si no se encuentra espacio para el registro en ninguna de las páginas asociadas se solicita la incorporación de una nueva página para acomodar el registro en esta, y el gestor de archivos devuelve al SGBD el RID del registro.
- Eliminar el registro *r* del archivo almacenado *a.*
- Actualizar el registro *r* del archivo almacenado *a*. El gestor de archivos tiene que recuperar la página que contiene el registro y después tratar de sustituir la información del registro por la nueva y acomodarlo en la misma página, si no es posible, lo ubica en otra página.

Debemos tener en cuenta las siguientes consideraciones para comprender mejor el gestor de archivos del SGBD:

- Organiza la BD en un conjunto de archivos del sistema operativo. Cuando crea uno normalmente solicita al gestor de disco la reserva de un conjunto de bloques grande aunque no se vaya a ocupar inicialmente. Esto lo hace para reducir la fragmentación externa y las peticiones de bloques del conjunto de bloques libres.
- El gestor de archivos puede solicitar al gestor de disco más bloques para un conjunto de bloques determinado (archivo para el SO) para satisfacer sus necesidades de almacenamiento.
- La liberación de bloques de un conjunto no se suele realizar por cuestiones de reorganización interna.
- Un conjunto de bloques del SO puede contener más de un archivo almacenado.
- Los registros almacenados integrantes de un archivo almacenado habrán de reorganizarse de forma que el gestor de archivos pueda recuperar todos ellos de forma secuencial. Para ello habrá de conocer a través de la cabecera del archivo almacenado la ubicación del primer registro y cada registro debe tener información para acceder al siguiente.
- El acceso directo a un registro se realiza a través del RID.

3.4. REPRESENTACIÓN DE LA BASE DE DATOS EN EL NIVEL INTERNO

La base de datos se representa de diferentes formas en los diferentes niveles de la arquitectura del SGBD, así que su representación a nivel interno no tiene por qué coincidir con su representación conceptual, que parece reflejar que cada conjunto de registros del mismo tipo es un fichero.

Como criterio básico, el SGBD en el nivel interno intentará *agrupar lo más cerca posible* los registros que pertenezcan al mismo conjunto de ítems a nivel conceptual o que pertenezcan al mismo archivo almacenado en el nivel interno, para que estos puedan recuperarse conjuntamente cuando se lea un bloque de datos del dispositivo de almacenamiento donde se encuentren.

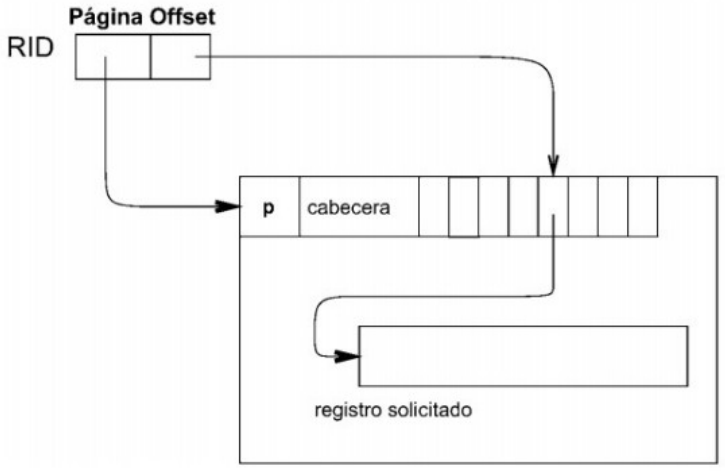
Podemos considerar que la BD en el nivel interno está constituida por un conjunto de páginas en las que se van ubicando los registros en el momento de su inserción. Con respecto a cómo se agrupan los registros en las páginas del nivel interno existen dos técnicas:

- **Agrupamiento intra-archivo:** consiste en ubicar en una página registros del mismo tipo. Es la más frecuente y la más eficiente, puesto que hay una elevada probabilidad de que, junto con un registro de un determinado tipo, se quiera recuperar también a sus predecesores o antecesores.
- **Agrupamiento inter-archivo:** ubica en la misma página registros procedentes de diferentes archivos. Para que resulte eficiente, debe existir una fuerte relación entre los registros, de forma que su recuperación conjunta sea una operación muy frecuente. Esto se da, por ejemplo, entre los registros que proceden de un conjunto de entidades fuerte y los conjuntos de entidades débiles asociados.

La descripción de la implementación del nivel interno es muy general y en cada SGBD comercial puede haber diferentes variantes. Por lo general, para localizar la primera página de cada conjunto de manera directa, la página 0 del conjunto de páginas va a almacenar dicha información, incluyendo también al conjunto de páginas libres.

Dentro de una página siempre se va a mantener el orden lógico de los registros, que estarán ordenados por su campos clave. Esto se consigue a base de desplazar si es necesario los registros existentes a la hora de ubicar uno nuevo.

Para, mediante consultas realizadas a nivel conceptual o externo, localizar un registro, tenemos el RID, que es el mismo durante toda la existencia del registro y permite acelerar este proceso enormemente. El RID tiene dos partes, la primera nos indica el número de página donde se encuentra dicho registro y la segunda nos indica el *offset* dentro de esa página. El direccionamiento indirecto a través del *offset* dentro de la página se lleva a cabo gracias a la existencia de un vector que se encuentra en la cabecera de esta y tiene tantas celdillas como registros pueden ubicarse en ella, conteniendo cada celdilla la dirección lógica de registro dentro de esa página.



Gracias a esta forma de acceso, los RIDs no se modifican nunca aunque se desplace un registro dentro de la página, ya que solo es necesario cambiar su dirección en la celdilla correspondiente de la cabecera. Esto es muy importante ya que el RID se usa en muchas estructuras asociadas (índices, tablas hash...) y esto consigue que su mantenimiento sea fácil.

Puede concluirse que no hay una relación directa fichero-almacenado/fichero-físico ya que cada conjunto de páginas irá almacenado en uno o varios ficheros ffísicos a nivel del sistema operativo.

3.5. MÉTODOS DE ORGANIZACIÓN Y ACCESO A LOS DATOS

Cuando tratamos con SGBD debemos asumir que los datos no caben en memoria principal, así que el almacenamiento externo será el soporte para cualquiera de las operaciones lógicas. Además se va a suponer que hay varios usuarios, un solo procesador, un controlador y un disco para las explicaciones.

De forma esquemática, los criterios de evaluación de la organización y acceso a los datos son:

- Tiempo de acceso a los datos, dominado por el coste de las operaciones de E/S y medido en número de registros accedidos bajo diferentes condiciones:
  - Por clave de búsqueda.
  - Por intervalo en la clave de búsqueda.
- Número de accesos a disco (ya sea para lectura o escritura) en las operaciones de inserción y de borrado de datos.

Ahora vamos a explicar los modos básicos de organización de archivos, sus mecanismos de acceso más utilizados y analizaremos los criterios de evaluación más relevantes.

3.5.1. Organización secuencial

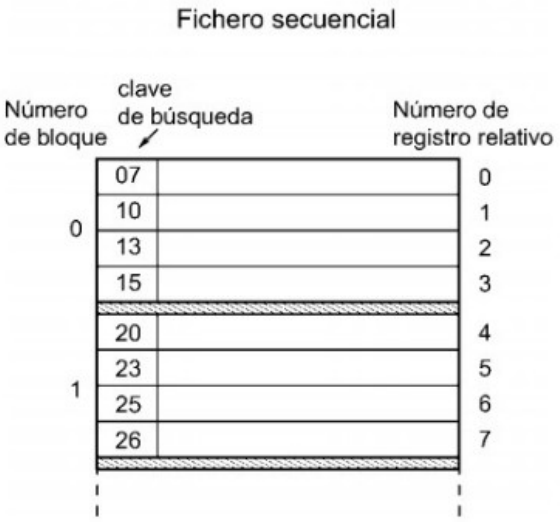
Los registros están almacenados consecutivamente en un dispositivo de almacenamiento, de forma que para acceder a un registros que esté en la posición 5 debemos pasar obligatoriamente por los 4 que le preceden.

Proceso de consulta

En el caso de acceso secuencial, una consulta implica recorrer uno tras otro cada uno de los registros comparando su clave con el valor que estamos buscando. El orden de eficiencia es *O(N)*. En una universidad que contenga unas decenas de departamentos no es preocupante siempre y cuando la búsqueda no se repita miles de veces, pero si se busca por alumos el tiempo de búsqueda puede llegar a ser mucho mayor, cosa inaceptable en una base de datos real.

En un fichero de acceso secuencial los registros están ordenados por una *clave fisica*, que es la que los mantiene ordenados dentro del fichero (clave de búsqueda en la figura).

La operación de inserción de un nuevo registro supone, en primer lugar, buscar el bloque que le corresponde. Si encuentra sitio se inserta, si no o se crea un nuevo bloque o un bloque de desbordamiento. Se suele dejar espacio vacío en los bloques para evitar problemas de reorganización desde el principio, aunque eso no implica que las reorganizaciones no vayan a tener lugar si la BD crece continuamente.



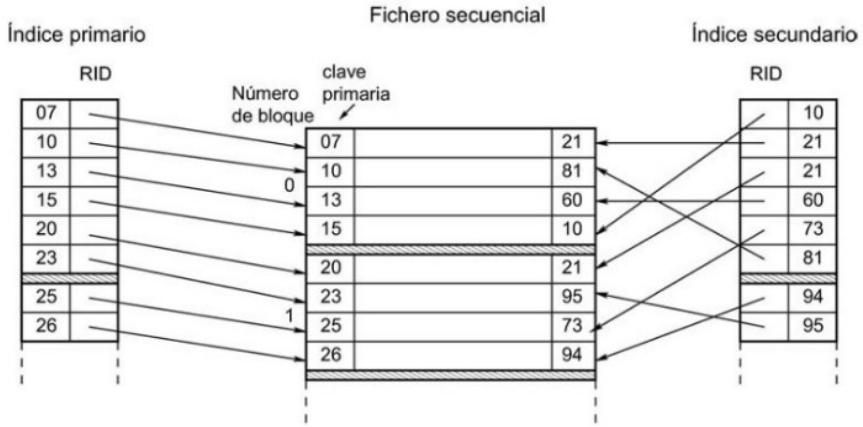
Para borrar un registro, una vez realizada la búsqueda del registro por valor de clave, el borrado puede implicar una reorganización local de los registros de un bloque.

Como podemos ver, esta forma de organizar los registros está llena de inconvenientes que podrían subsanarse usando estructuras adicionales que aceleren la localización de los datos y disminuyan el número de bloques de disco transferido, como la indexación.

3.5.2. Indexación

El método de búsqueda basado en un índice disminuye el tiempo de acceso a los datos por una clave de búsqueda. Se puede implementar de varias formas:

3.5.2.1. Fichero secuencial indexado



A parte de un fichero secuencial como forma de organización básica disponemos de una estructura adicional llamada *fichero índice* en el que sus registros poseen un campo denominado campo clave, por el que se realiza la búsqueda y un campo de referencia que contiene RIDs de registros. Vamos a suponer que la clave física es única. Si el índice que hemos considerado como clave de búsqueda también lo usamos como campo clave por el que ordenar el fichero de datos le llamamos *índice primario*.

Cuando se monta un índice primario sobre un fichero de datos se establece una correspondencia directa entre los valores del campo clave y el registro al que pertenece a través del campo referencia. Llamamos índices *densos* a aquellos en los que, como en la imagen, los registros del fichero de índice son más pequeños que los del fichero de datos aunque el número sea el mismo en ambos.

También se puede crear índices sobre otros campos que no sean la clave física del fichero de datos, los llamados *índices secundarios,* donde la clave de búsqueda se puede repetir. Listar de forma completa el fichero de datos usando el índice secundario es más lento que usando el primario

Proceso de consulta

Se hace una búsqueda del valor clave con ayuda del índice y se obtiene el RID del registro de requerido, lo que implica el recorrido secuencial del índice. Luego hay que hacer una lectura

adicional de disco para recuperar el bloque de datos donde se encuentra el registro buscado. Es una búsqueda más rápida que en el fichero de datos, sobre todo si el índice completo se puede almacenar en memoria principal.

Para buscar por intervalo se busca el primer registro en el índice y a partir de ahí se lee secuencialmente.

Para insertar un nuevo registro de datos supone las mismas operaciones de antes, solo que además hay que actualizar también el índice insertando una nueva entrada. Para el borrado habría que borrar la entrada del índice junto con lo que se hacía antes. Por esto las operaciones de borrado e inserción se ven ralentizadas.

3.5.2.2. Índices no densos

Por lo general los índices son muy grandes, por lo que se pueden mantener en memoria principal. Para reducir el tamaño se crearon los índices no densos, que contienen registros compuestos por la clave de búsqueda y la dirección de comienzo donde puede encontrarse el registro consultado, pero el número de registros en el índice se reduce al número de bloques del fichero de datos, de forma que el acceso secuencial al índice no denso es muy rápido. La búsqueda usando índices no densos difiere de la que usa un índice denso en que:

- Una vez encontrado el bloque donde podría estar el registro que se está consultando (se selecciona del índice la entrada inmediatamente inferior al valor clave para coger su RID) se carga el bloque entero y se hace ua búsqueda secuencial dentro del mismo para encontrar el registro concreto, que al estar ya en MP es más rápida.
- Cuando se realiza una búsqueda en el índice no denso no se tiene garantía de encontrar el registro hasta después de consultar el bloque de datos, es decir, hasta que no se consulta no se puede saber si un registro existe o no.

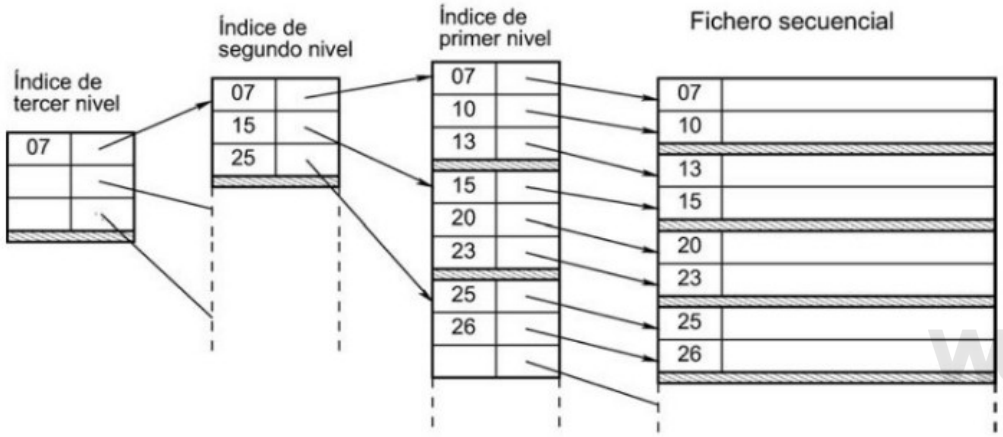
Solo puede haber índices no densos sobre la clave física, no puede haberlos sobre una secundaria.

Las inserciones y borrados son menos frecuentes, ya que solo ocurren cuando la operación afecta al valor más pequeño del bloque.

Por último, se puede montar un índice sobre más de un campo de un registro, esto es, sobre la concatenación de varios campos.

3.5.2.3. Índices jerárquicos o índices multinivel

Para disminuir el tamaño de los índices para que quepan en MP se planteó crear un índice de índices o índice multinivel, como en los sistemas de arhivos UNIX s5fs. Un índice así tiene, en primera instancia, un fichero índice sobre un fichero de datos de primer nivel. Sobre este, se puede crear un



segundo nivel, que será un índice primario y que se plantea como no denso y tendrá una entrada por cada bloque del segundo nivel. Esto se puede repetir añadiendo más niveles como en el ejemplo.

Estos índices reducen el número de accesos a disco cuando se busca un registro por valor de la clave de búsqueda del índice. Con un índice multinivel con *n* niveles como mucho se realizarán *n* accesos a disco (ya que el índice superior siempre estará en MP).

Estos índices multinivel tienen problemas con las inserciones y borrados sobre los índices, ya que todos ellos son ficheros ordenados. Para resolver esto a menudo se usan índices multinivel que dejan espacio en cada uno de sus bloques para insertar nuevas entradas, lo que se denomina un *índice multinivel dinámico* y se suele implementar con árboles B y árboles B+.

3.5.2.4. Indexación por árboles B

Los árboles-B (con B de *balanced*) son una generalización de un índice multinivel y un buen mecanismo para construir índices equilibrados. El bloque del nivel superior de un índice multinivel se puede considerar la raíz y los bloques del índice de nivel inferior las hojas. El propósito de este es restringir el proceso de búsqueda y minimizar el número de bloques consultados para localizar el registro deseado.

Cada nodo coincide con un bloque de memoria. Las claves contenidas en cada bloque de un nivel nos guían hasta el siguiente bloque del nivel inmediatamente inferior y así sucesivamente hasta llegar al bloque de fichero de datos que necesitamos.

Hay dos tipos de nodos, *nodos internos*, que constituyen el árbol en sí, y *nodos hoja*, que reciben el nombre de *conjunto secuencia* y proporcionan un acceso directo a los registros al ser un índice de primer nivel (denso o no) sobre el fichero de datos.

La estructura de un nodo interno de un árbol de orden *n* tiene, como mucho, *n - 1* valores de clave en el nodo (ya que el primero es el valor buscado en el padre, y no hace falta almacenarlo) y *n* punteros P_i que apuntan a un nodo hijo.

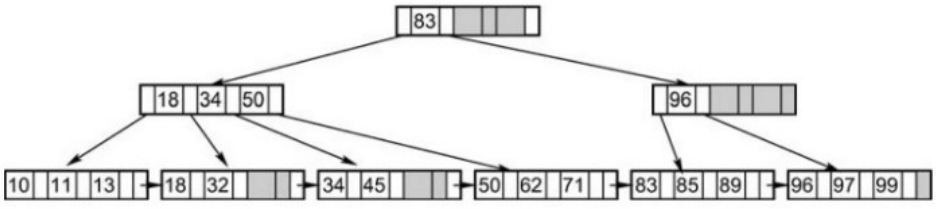


FIGURA 3.17 Ejemplo de árbol B

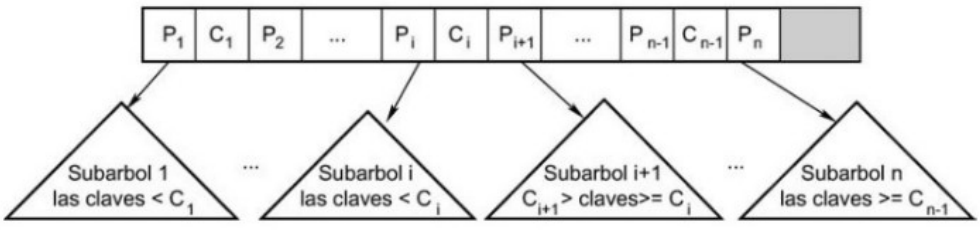


FIGURA 3.18 Nodo interno de un árbol-B

Se han de cumplir las siguientes restricciones dentro de los nodos:

1. Los valores de clave *C* están ordenados dentro del nodo según $C_1 < C_2 < ... C_m$, con $m \leq n-1$
2. Puede quedar espacio vacío dentro de los nodos, pero deben estar rellenos como mínimo a la mitad.
3. Para todos los valores *x* del subárbol apuntado por *Pi* se cumple que $C_{i-1} \leq x < C_i$, excepto para $i=1$ donde $x < C_1$, *y para* $i = m$, donde $x \geq C_m$.

La estructura de los nodos hoja es diferente a la de los nodos terminales de los índices multinivel.

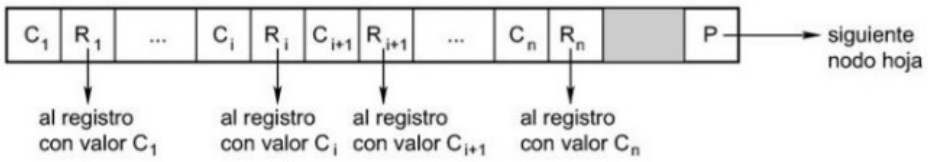


FIGURA 3.19 Estructura de un nodo hoja de un árbol-B

Están sujetos a las siguientes restricciones:

1. Las claves dentro de un nodo están ordenadas según $C_i < C_j$ para $i < j$.
2. Todas las claves de un nodo han de ser menores que las claves siguientes en el conjunto secuencia.
3. Los nodos han de estar como mínimo rellenos a la mitad.
4. Todos los nodos hoja están en el mismo nivel. Por esta restricción los árboles-B son árboles equilibrados.

Proceso de consulta

Para buscar un valor en el nodo raíz se busca el valor inmediatamente inferior y se sigue su puntero. Si no existe un valor como ese quiere decir que el puntero que hay que coger es el primero. Se sigue esto consecutivamente hasta encontrar la entrada en el nodo hoja del conjunto secuencia.

En una búsqueda por intervalo se hace lo mismo que una búsqueda directa, solo que cuando se llega al nodo hoja estos se pueden recorrer secuencialmente ya que cada nodo tiene un puntero al siguiente. En los índices densos los nodos hoja que forman parte del recorrido tienen las RIDs de los registros así que esta consulta es bastante eficiente. Las inserciones y los borrados el libro dice que son cosa de Estructuras de Datos y que no va a entrar ahí.

3.5.3. TÉCNICAS DE ACCESO DIRECTO

Buscamos, maś que una estructura adicional, un algoritmo que nos indique directamente la posición del registro en cuestión.

3.5.3.1. Ideas básicas acerca del acceso directo

Calculan la dirección de un registro mediante la aplicación, a un campo determinado del mismo que actúe como clave física, de algún algoritmo o función. Lo más sencillo es incluir como campo lógico del mismo la dirección que va a tener o una transformación muy simple de esta, por ejemplo,

para un conjunto de encuestas anónimas, numerarlas por orden de llegada y usar ese campo como clave física, cosa que es fácil de transformar en una dirección única. No siempre es posible establecer una clave física asociada a un campo que sea totalmente correlativa y única para cada registro, lo que se hace es encontrar un algoritmo que transforme los valores de un campo en número que usar para obtener el RID del registro.

El proceso de direccionamiento transforma la clave física k de un registro en un valor entero entre 0 y M, siendo M un valor estimado del total de registros que tendrá el fichero. Posteriormente, se sitúa el registro en el fichero. Este mismo algoritmo de direccionamiento se usa para recuperar los registros a partir de la clave.

Con este tipo de almacenamiento los registros no están almacenados secuencialmente, de forma que esta organización no es muy adecuada para recuperar registros por intervalos.

3.5.3.2. Los algoritmos de direccionamiento

Una vez convertida la clave en un valor numérico se le asocia un valor entero positivo en un rango de 0 a M. Este algoritmo suele estar basado en un mecanismo de generación de pseudoaleatorios, y entre los más comunes están:

- **Cuadrados centrales:** se eleva la clave al cuadrado y se eligen tantos dígitos del centro de esta como sea necesario.
- **Congruencias:** se divide la clave por M y se toma el resto.
- **Desplazamiento:** se superponen adecuadamente los dígitos binarios de la clave y luego se suman. Es muy adecuado para claves alfanuméricas.
- **Conversión de base:** se cambia el sistema de numeración y se suprimen algunos dígitos resultantes.

3.5.3.3. Problemas con el acceso directo

A menos que se diseñe un campo como en el ejemplo de las encuestas es prácticamente imposible encontrar una transformación asociada a un campo que dé un valor entero positivo en un rango de valores limitado tal que:

1. No haya dos valores distintos del campo clave que den lugar al mismo número, es decir, que evite *colisiones* o no produzca *sinónimos*.
2. Que no produzca muchos huecos, es decir, que no se acumulen todos los valores en una determinada zona del intervalo dejando vacío l resto. Este problema está muy relacionado con el anterior.

Para evitar las colisiones se suele combinar el acceso directo con una gestión mediante listas. Se asigna al fichero una zona de desbordamiento, y cuando se produce una colisión, el registro problemático se almacena en la zona de desbordamiento manteniendo conectados los sinónimos mediante una lista. Si hay muchos sinónimos puede ser casi imposible mantener este sistema, ya que habrá una gran cantidad de listas que mantener y el espacio de desbordamiento puede ser mayor que el original. Esto ha dado lugar a que no se use el acceso directo puro sino más bien el hashing.

3.5.3.4. El hashing básico o estático

La idea básica del hashing es que si el problema principal es que los valores de las claves no están uniformemente distribuidos en el intervalo [0,M]. Para realizar esto lo que se hace es dividir el espacio del fichero en *cubos*, de forma que el algoritmo de direccionamiento no sitúa al registro en una posición del fichero sino en un cubo. Se diseña el algoritmo de forma que ciertos rangos de valores tengan asignados más cubos que otros, y se aportan además *cubos de desbordamiento* para situar aquellos registros que no caben en sus cubos asignados.

Para insertar un registro en un fichero haremos lo siguiente:

1. Transformar la clave.
2. Localizar el cubo correspondiente.
3. Si hay sitio se inserta el registro y hemos terminado.
4. Si no hay sitio, se sitúa el registro en un cubo de desbordamiento conectándolo con el cubo que realmente le corresponde mediante punteros.

El proceso de búsqueda sería:

1. Transformar la clave.
2. Localizar el cubo correspondiente.
3. Realizar una búsqueda secuencial dentro del cubo.
4. Si hemos encontrado el registro, el proceso termina.
5. En caso contrario, se impone un barrido por punteros a través de los cubos de desbordamiento.

Con este planteamiento, los elementos a tener en cuenta para diseñar una organización de hashing son:

- El número de cubos.
- El tamaño de los cubos, es decir, el número de registros por cubo. Es importante tener en cuenta la relación de este tamaño con el de los bloques, ya que no tiene sentido hacer accesos a disco de más para traer un cubo, lo ideal sería que un acceso trajera un cubo entero.
- La transformación clave/dirección, que debe tener en cuenta la distribución de la clave segń rangos para que no se llenen mucho unos cubos y otros se queden vacíos.
- Cuántos cubos de desbordamiento va a haber y cómo se van a distribuir.

3.5.4. EL HASHING DINÁMICO

El hashing estático sigue teniendo problemas, que se centran en el hecho de que es necesario conocer la distribución previa de las claves, ya que en caso contrario puede ocurrir que se asignen a un rango de valores menos cubos de los necesarios que lleven a problemas de colisiones, lentitud por la gestión del desbordamiento y frecuentes reorganizaciones y cambios de algoritmo.

La solución es no tener que hacer esa evaluación previa y trabajar de forma dinámica, de manera que, partiendo de una configuración uniforme y de pocos cubos, los restantes se vayan generando conforme se vayan necesitando y se asocien al rango de valores donde sea necesario.

La idea básica es considerar que el valor transformado del campo clave nos lleva a la entrada de una tabla índice que se almacena en memoria y la salida de esta es la dirección del cubo donde se encuentran los registros que tienen asociado este valor transformado. Puede ocurrir que varias entradas conduzcan al mismo cubo (pasa al principio), pero a menudo que se van insertando nuevos registros se van generando nuevos cubos y cambiando las salidas de la tabla índice.

3.5.4.1. Algoritmo de hashing dinámico

Datos de partida:

- *k* que es el campo clave física del registro asociado al fichero que vamos a organizar según esta estructura.
- $k' = h(k)$ es un valor entero entre 0 y *M*, salida del algoritmo aplicado.
- *n* es el número de bits que tiene *k'* en binario
- Un valor entero *d* tal que $0 < d \leq n$, los *d* primeros dígitos de *k'* seleccionarán el cubo, de manera que todos los registros cuyo valor de *k'* pasado a binario tengan sus *d* primeros dígitos iguales estarán en el mismo cubo. *d* es la pseudoclave.
- Un valor entero *b* tal que $b < d \leq n$. Inicialmente, el archivo tiene 2^b cubos distintos y como máximo tendrá 2^d.

Algoritmo:

- Se considera una tabla índice en memoria con 2^d filas. En la primera columna de esta tabla (valores de campo clave) se sitúan todas las posibles sucesiones de *d* dígitos binarios. *d* es la profundidad de la tabla.
- En principio, todas las entradas cuyos *b* primeros dígitos son iguales apuntan al mismo cubo. Allí se almacenan los registros cuyo valor de *k'* en binario tiene esos *b* primeros dígitos. Todos los cubos tienen en principio profundidad local igual a *b*.
- Cuando se llena un cubo se divide en dos, poniendo en uno de ellos los registros con el dígito $b + 1$ de *k'* a 0 y en otro los que lo tienen a 1. La profundidad local de estos cubos aumenta en una unidad. Si un cubo de los generados anteriormente se vuelve a llenar, se divide en dos de nuevo con el mismo proceso pero con el dígito $b + 2$.

Se puede ver que el hashing dinámico supera los problemas clásicos del acceso directo ya que se vana producir menos colisiones y no van a aparecer demasiados huecos, pero como conveniente tenemos que hay que usar una tabla índice adicional que implica nuevos accesos a memoria si esta no se mantiene en MP mientras se trabaja del fichero, pero como esta depende de *d*, si se ajustan bien los parámetros *b* y *d* esta no tiene por qué ser inmanejable.

3.5.5. OTRAS TÉCNICAS: ESTRUCTURAS MULTILISTA

3.5.5.1. Ideas básicas de la organización encadenada

Esta se orientó inicialmente para acceder a los registros a según un campo que no sea la clave física. La idea básica es que se construyen listas lineales enlazando aquellos registros que tienen el mismo valor en el campo clave y el sistema se complementa con un fichero adicional donde se recogen los valores de la clave convenientemente ordenados y conectados con el principio y el final de su lista.

Estas estructuras se denominan también estructuras 'padre/hijo', considerándose que el archivo padre es el de los valores del campo clave. Se puede montar una estructura con más de un padre para el mismo archivo hijo.

Ventajas e inconvenientes

Sus ventajas son que:

- Es muy eficaz para poder acceder según los valores del campo clave. De hecho, con una buena estructura de agrupamiento, es uno de los sistemas de acceso más rápido.
- Ocupará menos memoria que una tabla índice similar, ya que los datos aparecen una sola vez. Se elimina la redundancia y se facilitan las operaciones de actualización.

Presenta los siguientes inconvenientes:

- Cuando se desea obtener la información del fichero padre a través del hijo el aceso puede ser muy costoso.
- Una estructura tan compleja, con tantas cadenas de punteros implicadas, suele tener problemas de mantenimiento.
- Siempre hay que diseñar un sistema de acceso alternativo para todos los ficheros implicados, ya que el proceso siempre empieza con un acceso al padre o al hijo.

Para evitarlos se han propuesto variantes como usar cadenas de punteros dobles o mantener los registros completos en el fichero hijo, de forma que los accesos hijo/padre sean innecesarios.

3.5.5.2. La estructura multilista y la conexión entre ficheros

Si solo se usaran las estructuras multilista para recuperar registros a partir de los valores de un campo que no sea su clave física, esta organización no sería muy útil, ya que esto lo resuelve la indexación por claves secundarias con menos complejidad. Su importancia radica en que puede conectar ficheros a través de un *campo común*.

Si suponemos que los ficheros padre e hijo tienen su propia información asociada y que se conectan porque el hijo incluye un campo que referencia a los registros del padre, tendremos los dos ficheros conectados a través de este campo.

Esta organización nos permite reflejar algunas de las características que corresponden al concepto de base de datos, ya que conectan distintos ficheros de forma que la información no sea redundante y se pueda recuperar moviéndose a través de los mismos.

El uso de este tipo de organización fue el origen de los primeros modelos de bases de datos, y la estructura lógica de estos es la de los modelos de datos basados en grafos ya estudiada.