BASES DE DATOS

Guión Tema 8

Organización Física de los Datos
Gestión de Ficheros

Profesor : Héctor Gómez Gauchía

ORGANIZACIÓN FÍSICA DE LOS DATOS. GESTIÓN DE FICHEROS

1 Introducción

En los temas anteriores: hemos visto el nivel conceptual o lógico de las bases de datos.

- Una base de datos en el modelo relacional se ha considerado una colección de tablas (nivel conceptual o lógico facilitar el diseño) No se carga al usuario con
- Los detalles físicos de la implementación del sistema: *ficheros* o *archivos*, que incluye
- El *rendimiento* (p.e. tiempos de respuesta para una consulta), que depende de la *eficiencia* de
- La *estructura de datos* que se utiliza para representar la información de la base de datos y de
- La eficiencia del *sistema de ficheros* para manejar estas estructuras de datos, así como de
- La velocidad y tipo de los *dispositivos de almacenamiento* de datos: volátiles y persistentes

En este tema se ven estos aspectos de la *implementación* de los modelos de datos lógicos y se definen las *estructuras de datos* que permitirán un acceso rápido a los datos:

- Varias estructuras de organización alternativas, cada una de las cuales es apropiada para un tipo distinto de acceso a datos y habrá que llegar a
- Un compromiso entre la eficiencia de los distintos tipos de operaciones de acceso a datos.
- La elección final de la estructura de datos se hará teniendo en cuenta el uso esperado del sistema y las características físicas de la máquina.

Cuatro conceptos clave:

- **Campo:** Elemento básico de datos que contiene un valor único.
  Ejemplos: Nombre, Primer apellido, Segundo apellido
  Caracterizado por su tamaño y tipo. El tamaño puede ser fijo o variable.
- **Registro:** Colección de campos relacionados que se tratan como unidad en programas.
  Ejemplo: Registro Persona, compuesto por los campos Nombre, apellido 1º, apellido 2º
- **Fichero o archivo:** es una colección de registros similares.
  Entidad única para los programas y usuarios. Nombre único, pero pueden tener alias.
- **Base de datos:** Colección de datos relacionados.p.e.: tablas relacionadas entre sí
  Implementada con uno o varios ficheros del mismo o diferente tipo.

Como parte de la implementación física de una BD se encuentran varios *tipos de ficheros*:
(en algunos gestores están agrupados en un solo fichero)

- Ficheros de datos, que almacenan la base de datos
- Ficheros del diccionario de datos, con información acerca de la estructura de la B.D.
- Ficheros de índices, que permiten acceso rápido a los datos (de la BD y del diccionario).

Cuáles son las *necesidades básicas* que motivan la existencia de ficheros:

1. Almacenar gran cantidad de información que no cabe en la memoria principal
2. Persistencia de la información más allá de distintas ejecuciones
3. Acceso a información de forma concurrente

El responsable es el SO y lo realiza mediante la gestión del *Sistema de Ficheros*. Esto da dos puntos de vista a estudiar:

1. El usuario (o programa) como cliente del sistema de ficheros
2. El SO como gestor del sistema de ficheros da servicio al SGBD

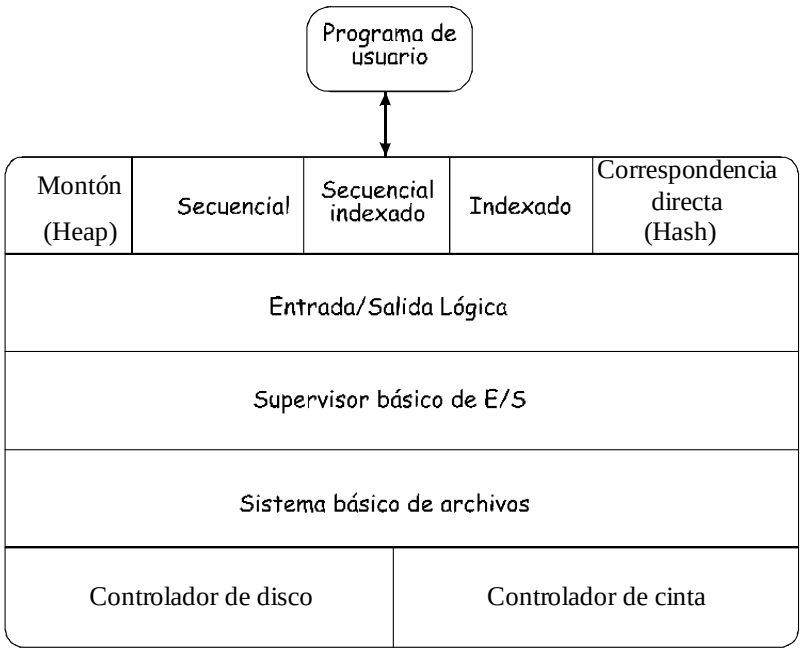
2 Gestor del Sistema de Ficheros

Es el software que proporciona a los usuarios servicios relativos al uso de ficheros.

2.1 Servicios básicos de los Sistemas de Ficheros

1. Capacidad para cumplir con las operaciones básicas de ficheros:
   1.1 Crear, borrar y modificar ficheros
   1.2 Acceso controlado a los ficheros de otros usuarios
   1.3 Reestructuración adecuada de los ficheros
   1.4 Movimiento de datos dentro del fichero
   1.5 Copias de seguridad y recuperación
   1.6 Acceso mediante nombres simbólicos
2. Garantizar la validez de los datos.
3. Optimizar el rendimiento [productividad + tiempos de acceso]
4. Ofrecer soporte a los diversos tipos de dispositivos de almacenamiento.
5. Garantizar la fiabilidad [minimizar pérdidas o destrucción de datos]
6. Ofrecer un conjunto estándar de rutinas de entrada/salida.
7. [Sistemas multiusuario] Proporcionar acceso múltiple.

2.2 Arquitectura de los Sistemas de Ficheros



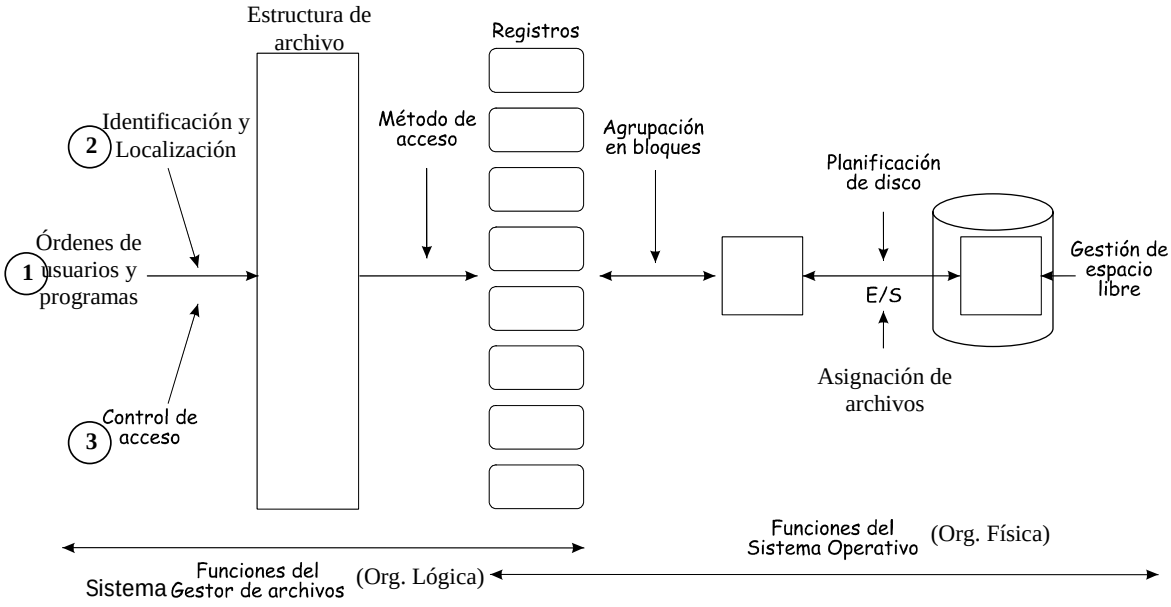
- Controladores (software) de dispositivos: Comunicación entre el controlador hardware del dispositivo y el sistema operativo. (drivers)
- Sistema básico de ficheros (Nivel físico de E/S): Trata con las transferencias de bloques de datos entre el almacenamiento primario (memoria RAM) y secundario.
- Supervisor básico de E/S: Inicio y terminación de las operaciones de E/S. Realiza la selección del dispositivo dependiendo del fichero requerido. Planifica las operaciones para mejorar el rendimiento. Se asignan los buffers de E/S y la memoria secundaria.
- E/S Lógica: Permite el acceso a los registros. El que hacen nuestros programas según el...
- Método de acceso: Interfaz estándar entre las aplicaciones y el sistema de ficheros.

2.3 Funciones de los Sistemas de Ficheros

Esquema básico de interacción con el sistema de ficheros:

1. Usuario o programa demanda una operación básica (eliminación, movimiento
2. El sistema de ficheros identifica y ubica el fichero en cuestión
3. El sistema verifica los permisos sobre el ficheros

La operación se solicita a nivel de registros (org. lógica) que no es la del sistema de E/S. (organización física).



3 Dispositivos Hardware para almacenar datos

Según la velocidad de acceso a los datos, el coste y la fiabilidad hay tipos de almacenamiento:

- **Almacenamiento primario: memoria volátil (caché y RAM)**

Memoria caché: rápida y costosa. Tamaño pequeño. Gestionada por el SO. No para usuario.

Memoria principal: es rápida (<100ns), barata, volátil, para datos en proceso

- **Almacenamiento secundario: discos magnéticos, memoria no volátil (*flash*),discos RAID**

Memoria flash: sólo de lectura programable y borrable eléctricamente

(Electrically Erasable Programmable Read-Only Memory, EEPROM). *No volátil*.
Velocidad lectura: igual que la memoria principal
Sucesivas escrituras: hay que borrar antes lo que había; por bancos completo.
Inconveniente: número limitado de ciclos de borrado (10 M).
Para dispositivos móviles, tarjetas inteligentes.

Discos magnéticos: La unidad principal más usada (ver sección siguiente).

Discos RAID. disposición redundante de discos independientes (RAIDs - Redundant Array of Independent Disks). Objetivo:

Mejora de la fiabilidad:
- redundancia. Creación de imágenes o sombras (mirrow).
- cada proceso de escritura se lleva a cabo en dos discos físicos

Mejora del rendimiento:
- distribuir la carga entre todos los discos: las cabezas lectoras se reparten el trabajo
- accesos paralelos en peticiones de datos masivas: recuperan bits desde discos diferentes

- **Almacenamiento terciario: discos ópticos y cintas** se extraen del dispositivo. Usos:

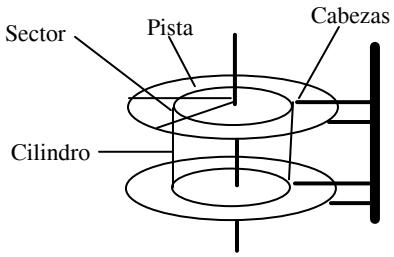
- copias de seguridad (backup)
- almacenamiento de datos poco usados y/o estables. ej.: datos históricos
- transferir información sin conexión ej.: programas

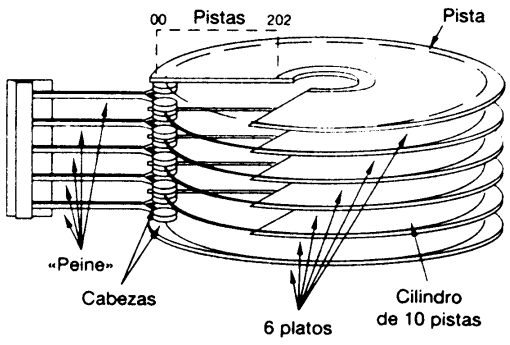
Almacenamiento óptico. Memoria sólo de lectura en disco compacto (Compact-Disk Read-Only Memory, CD-ROM). Memoria de escritura única y lectura múltiple (Write-Once, Read-only Memory, WORM). Dispositivos de almacenamiento combinados magnetoópticos que utilizan medios ópticos para leer datos codificados magnéticamente: Jukebox, DVD. (15GB)

Cintas. Son DAT, graban datos en formato digital. Son muy baratas y fiables.

3.1 Tipo más usado : discos

- Cada plato /disco tiene dos caras
- Cabezas de lectura escritura: una por cada cara de cada disco. Optimizar su recorrido
- La superficie del disco se divide en *pistas*, que se subdividen en *sectores*.
- Pista: división circular concéntrica de la superficie de un disco. Número fijo por disco.
- Sector: subdivisión de información dentro de una pista. Continene bloques enteros.
- Cilindro: conjunto de pistas de todas las caras que pueden accederse sin mover la cabeza.
- Bloque: Unidad mínima de manejo por el S. De Ficheros. En un bloque caben **n** *registros*.
- Un *sector* es la unidad mínima de información que puede leerse o escribirse en el disco.
- Cada sector puede tener más de un *bloque*, se crean al *formatear* el disco.





- Ejemplos de cálculo de espacio :

  Tamaños:

  8 bits = 1 byte; 1024 bytes = 1 kb; 1024 kb = 1 Mb; 1024 Mb = Gb; 1024 Gb = 1 Tb; 1024Tb=Pt

  Los sectores son muy variados, desde los 32 hasta los 4096 bytes; generalmente son de 512 bytes.

  4-32 sectores por pista.

  20-1.500 pistas en la superficie del disco.

  512*4*20*2(caras)=81.920 bytes/plato

  512*32*1500*2=49.152.000 bytes/plato

  9 Gb = 183 platos

Planificación de accesos a los bloques para minimizar el movimiento del brazo del disco. Algoritmo del ascensor: lee las peticiones de las pistas que encuentra en su camino (reordena las peticiones).

Algoritmo del ascensor: reordena las peticiones según la cercanía física entre sí de los lugares de acceso (pista, cilindro) pedidos y después ejecuta las peticiones secuencialmente que encuentra en el recorrido a modo de ascensor que para en todos los pisos seleccionados.

Memoria intermedia de escritura no volátil. El rendimiento de las aplicaciones de bases de datos, como los sistemas de procesamiento de transacciones, dependen mucho de la velocidad de escritura en el disco. Se usa RAM no volátil para acelerar la escritura en el disco de manera drástica. (RAM alimentada por baterías)

4 Organización lógica de ficheros y métodos de acceso a fichero

Cada técnica de indexación o alternativamente las técnicas hashing (funciones de asociatividad) se deben evaluar en base a:

- tiempo de acceso para encontrar un dato,
- tiempo de inserción (tiempo de buscar lugar adecuado + inserción + actualización de la estructura de indexación),
- tiempo de eliminación, espacio extra que ocupa la estructura.

4.1 Ficheros secuenciales

- Es la forma más común de organización de ficheros.

- Se caracteriza un campo único de cada registro (normalmente el primero) como **clave** para identificar el registro.

- Para la actualización y recuperación es una organización poco eficiente, aunque las técnicas de búsqueda se pueden mejorar llevando partes del fichero a memoria principal.

clave | campos...

4.1.1 Organización física

Los registros se almacenarán en bloques sucesivos según su orden de llegada. *Asignación de bloques*:

- Contigua (poco usada)

- Con lista enlazada : los bloques se enlazan entre sí

- Con lista enlazada con índice: como la anterior, con una tabla aparte con direcciones de bloques usados.

4.1.2 Operaciones básicas

- *Inserción* de un registro (ALGORITMO):

Cada registro nuevo se pone al final del fichero:
si caben, en el último bloque
si no: se pide un nuevo bloque y
(lista enlazada) se enlaza con el último, se coloca el registro, se actualiza la marca EOF.
(lista con índice) se añade a la tabla de direcciones, se coloca el registro.

- *Borrado* de un registro: bit de borrado en cada registro de un bloque con compactación periódica.

- *Búsqueda* de un registro: es exhaustiva desde el primer registro porque no hay ningún orden entre ellos.

4.1.3 Ficheros secuenciales ordenados : al crear registros se ordenan según su clave.

- *Insercción* más lenta al ordenar. Búsqueda binaria mas rápida: $\log_2$ (num.bloques)

- En BD solo se usan junto con un índice (los veremos) y se llaman *secuenciales indexados*.

4.2 Ficheros con organización o asociación directa (*hash*). Direccionamiento calculado

Aprovechan la capacidad de los discos para acceder a cualquier posición de dirección conocida.

Se requiere un campo clave en cada registro del fichero.

Los ficheros hash son estructuras ordenadas por claves de búsqueda.

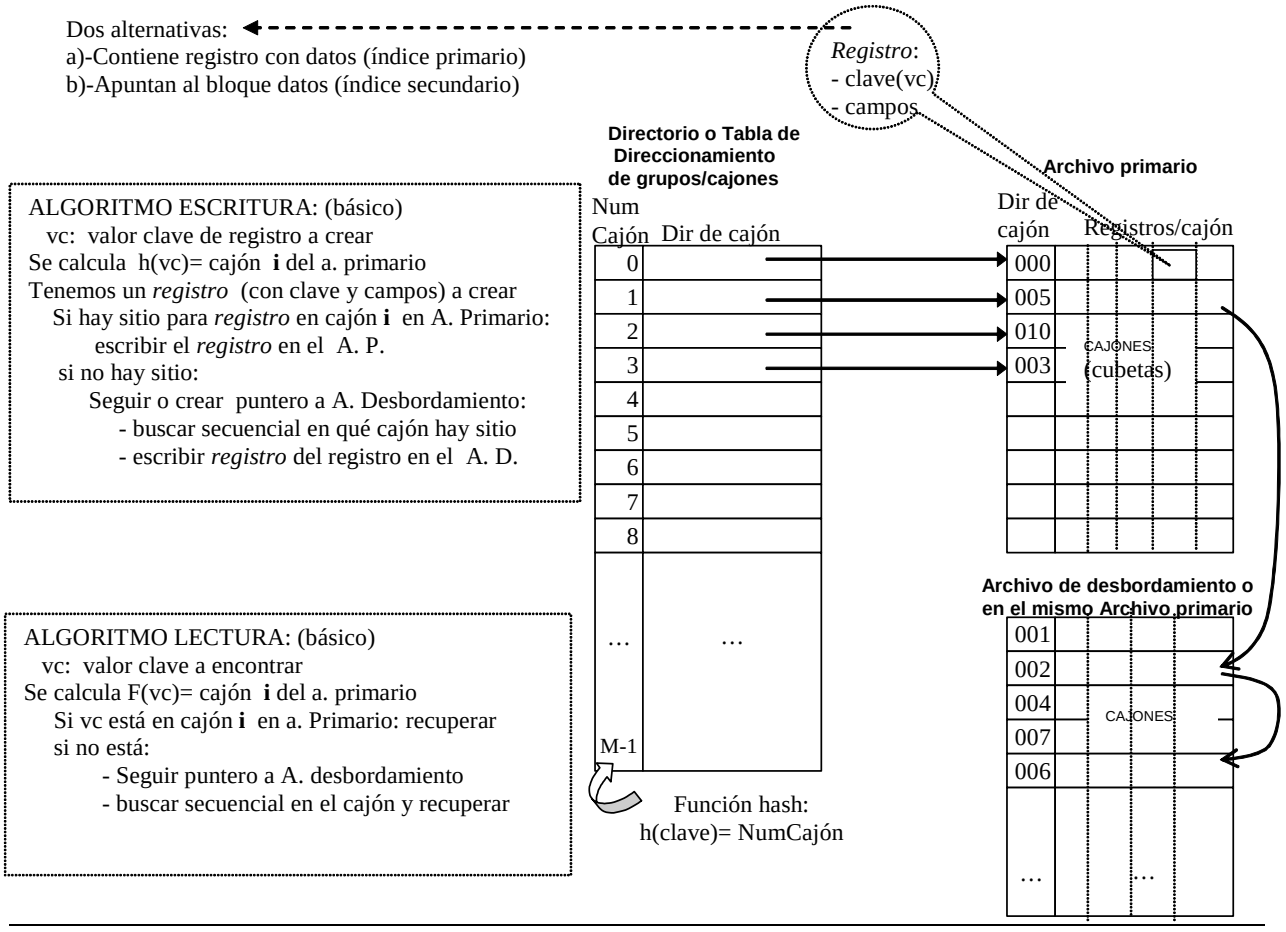
La clave de búsqueda: es el campo/s (atributos) que se usan para encontrar un registro(del fich.)

Para saber donde está el registro se usa una función de la clave f: V -> B, siendo V el conjunto de valores de clave y B el conjunto de cajones que almacenan registros.

Función de muchos a uno: un cajón o cubeta contiene registros con varios valores de la clave

EJ: $h(K) = K * mod M$, K es valor de clave y con M cajones, h(K) da valores entre 0 y M-1

Tamaño habitual de un cajón es de un bloque para optimizar los accesos.

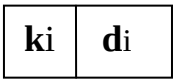


Gestión de cajones: El número de cajones M se fija a priori.

Colisiones: cuando la función hash de una clave da un cajón lleno se enlaza con otro nuevo en el desbordamiento.

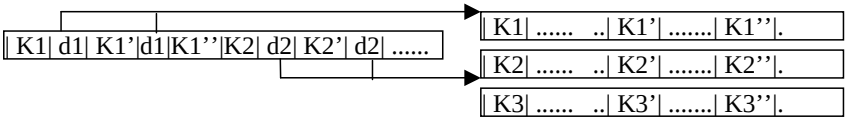
4.3 Ficheros con índices: Tipos de índices (Korth cap. 11)

Cada fichero tiene otro fichero con los índices, cuyos registros tienen el valor una clave y otro campo para la dirección de bloque donde está el primer registro de datos con el valor de la clave. Hay otras organizaciones que difieren ligeramente como los índices multinivel

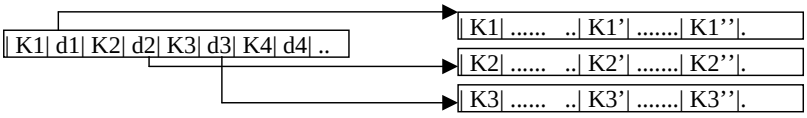


4.3.1 Tipos de índices según su estructura

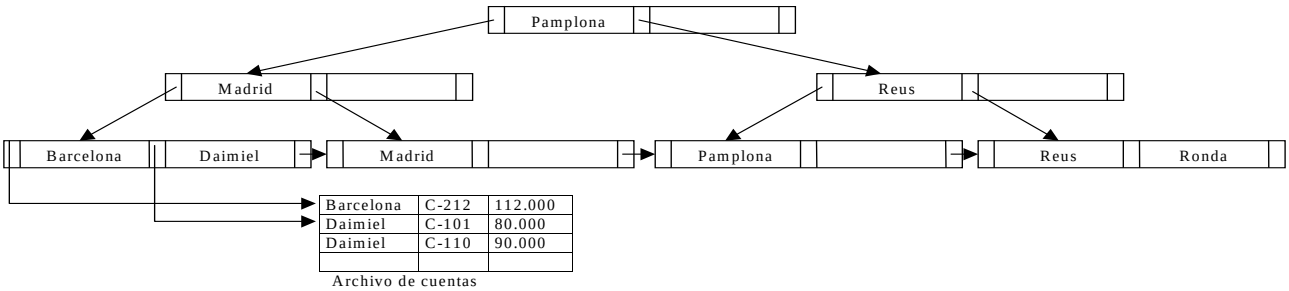
Índice denso o *Exahustivo* (dense): Aparece una entrada en el índice por cada valor de la clave de búsqueda. Para buscar un registro, se ubica en el primer registro con el valor del campo clave buscado y se procesa secuencialmente (puede haber más de un registro con el valor de la clave de búsqueda, sólo si coincide con una clave candidata habrá un único registro). Ventaja: Rapidez en la consulta. Inconvenientes: Espacio usado y menor rapidez en las actualizaciones.



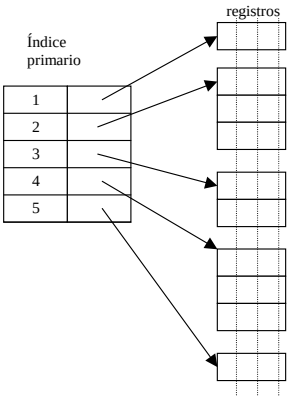
Índice disperso (sparse): No aparecen todos los valores de la clave de búsqueda. El acceso en lectura será en general más lento porque probablemente en el índice no se encuentre el valor del campo clave (hay que acceder al valor anterior y procesar secuencialmente en el orden de la clave). Esto se puede usar sólo si los registros están ordenados en esta clave de búsqueda.



Índices multinivel. Aparecen como consecuencia de los índices de gran tamaño que no pueden almacenarse en memoria y que implican un gran tiempo de acceso (incluso efectuando una búsqueda binaria en el fichero índice). Se crea un índice disperso sobre el índice denso. Se puede repetir el proceso tantas veces como sea necesario, creando una capa nueva con un índice disperso sobre el último índice creado. Un caso particular es el índice en árbol equilibrado B+, que se ve después.



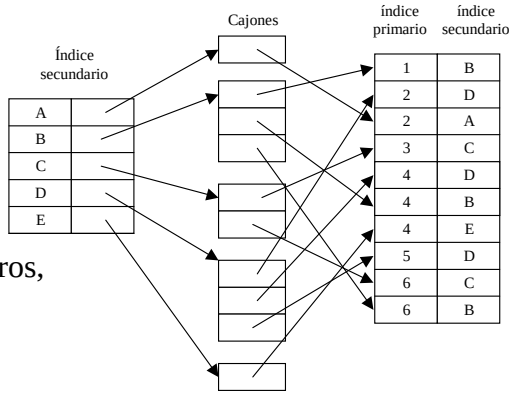
4.3.2 Tipos de índices según el uso y el orden de sus registros



Índices primarios

- Primario si asume una *ordenación* de registros en el disco por un campo o varios
- Estos componen la *clave de búsqueda*
- Puede haber varios registros con el mismo valor de la clave (NO clave candidata).

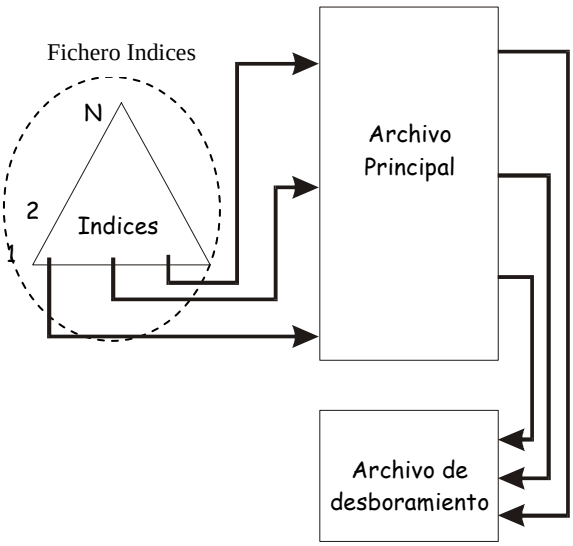
- Los registros del fichero índice tienen 2 campos:
  - El valor de la clave
  - La dirección del bloque de datos que contiene el primer registro de datos con esa clave.



Índices secundarios

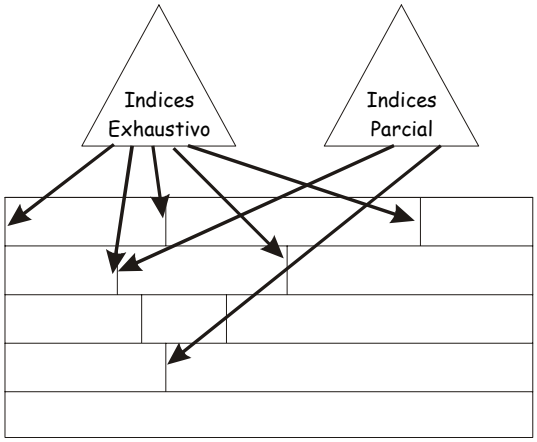
- Para acceder a los registros por otros campos.
- Los registros no están ordenados por estos índices**.**
- Deben ser índices densos, con entradas a *todos* los registros, usando doble indirección: cada valor del índice apunta a un cajón (bucket) con las dirección de todos los registros con ese valor de índice.

4.4 Ficheros con índices: secuenciales indexados



- Organización y asignación secuencial (no ordenada): todo el fichero es un gran espacio en disco con otro de desbordamiento. Lo nuevo es que tiene un:
- Con un índice (o varios) para acceso aleatorios.
- Así reduce los tiempos de acceso
- con más niveles de indexación,
  más complicación de gestión
  más eficiencia en los acceso.
- Limitación: un único campo clave del fichero.

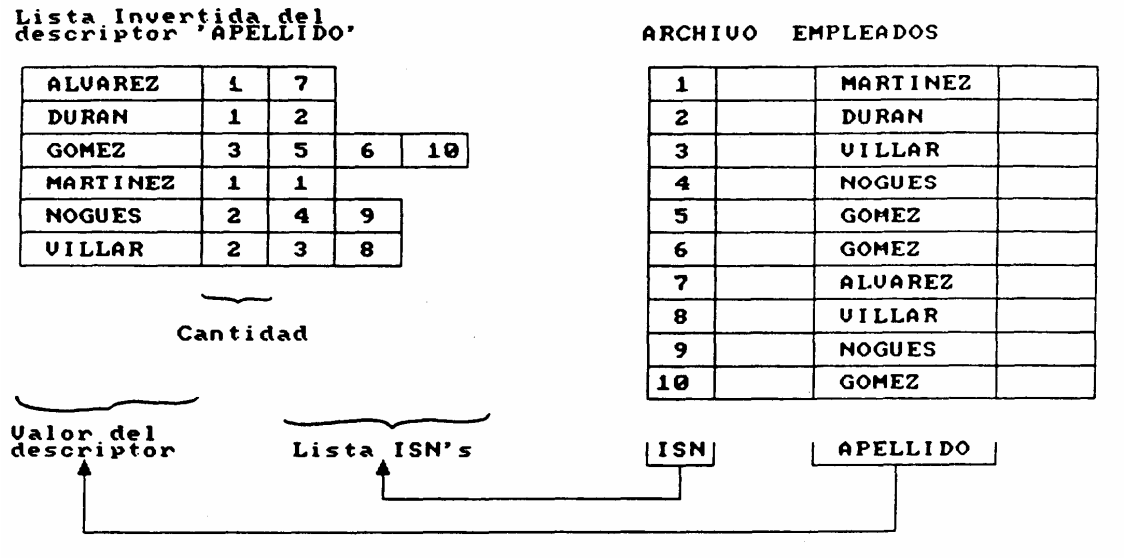
4.5 Ficheros con índices: indexados



- El fichero organizado por bloques.
- Con índices múltiples, uno por campo de búsqueda.
- Acceso a los registros en función de sus índices.
- Dos tipos de índices: denso y disperso
- Problema al crear registro:
  actualizar todos los índices es costoso.

4.6 Ficheros con índices:Ficheros invertidos (con tabla invertida)

- Con una clave de búsqueda.
- Estructura Lista invertida: - el número de registros con un valor de clave
  - ubicaciones de cada uno de ellos
- Se usan en varios SGBD, por ejemplo, ADABAS



---------- Vamos a ver en detalle una situación de las que se pueden dar ---------

4.7 *Organización física* de ficheros indexados con índice primario disperso (o escaso)

Es un ejemplo de organización física. El índice denso es fácil de hacer a partir de este.

- *Fichero de Datos* (H7P3)

Los bloques del fichero de datos son B1, B2,...., Bn. La asignación suponemos que es enlazada, es decir, los bloques B1, ..., Bn están enlazados.

El fichero de datos está ordenado secuencialmente por la clave de búsqueda del índice.

Los registros de cada bloque preceden a los del bloque siguiente.

Dentro de cada bloque los registros están ordenados de menor a mayor valor de la clave de búsqueda.

En la cabecera de cada bloque hay bits de lleno/vacío de bloque.

- *Fichero de índices*

Se crea un índice disperso con un registro índice <k,d>, clave y dirección, por *cada bloque* del fichero de datos. El fichero índice también se mantiene ordenado.

Operaciones:

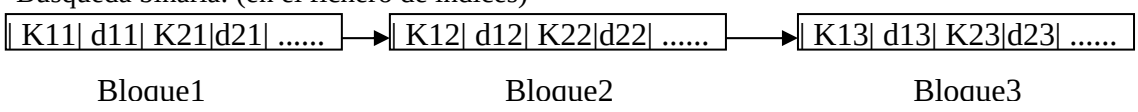
Definición: se dice que k' cubre a k si <k',d> es un registro del índice tal que k'≤ k y ocurre una de las dos cosas siguientes: o es el último registro del índice, o el siguiente registro <k'',d'> cumple k''>k.

- *Búsqueda en el fichero índice de una clave k' que cubra a k*

Es la clave que va a permitir encontrar los registros (puede haber varios !!) con clave k.

A) Búsqueda lineal: buscamos secuencialmente en los bloques del fichero índice. Eficiencia: se mide el nº de bloques que hay que llevar a memoria. En el caso peor hay que llevar todos. Es apropiado para índices pequeños.

B) Búsqueda binaria: (en el fichero de índices)



Se busca kc que cubra a k entre sólo las primeras de cada bloque del fichero índice.

Si K1j cubre a k (sólo entre las primeras de cada bloque) → se hace búsqueda secuencial en ese bloque de k' que cubre a k.

Búsqueda de un registro de datos con clave k (H7P3.a.)

1. buscar en el índice un registro <k',d> t.q. k' cubra a k. (es la operación anterior).
2. buscar el registro k en el bloque del fichero de datos con dirección d.

Modificación de un registro de datos con clave k

1. buscar el registro de datos con clave k (operación anterior).
2. para modificar la clave: borrar + insertar.
3. para modificar otro campo: recopiar el registro.

Inserción de un registro de datos con clave k

1. buscar el registro de datos con clave k para encontrar el bloque Bi dónde debe almacenarse.
2. Si Bi no está lleno:

   Colocar el registro en su lugar, desplazando a la derecha los registros con clave mayor (no se puede desbordar porque insertamos sólo 1 y sabemos que cabía). Modificar los bits lleno/vacío.
3. Si Bi está lleno:
   a. Si Bi+1 no está lleno se usa para desplazar el último de Bi al primero de Bi+1. Modificar el fichero índice → añadir un registro con la clave del primer registro del bloque Bi+1.
   b. Si Bi+1 está lleno, pedir un nuevo bloque que se usa para el desplazamiento. Insertar un nuevo registro en el índice. Actualizar lleno/vacío. Encadenar el nuevo bloque detrás de Bi y delante de Bi+1. (H7P3.b.)

Caso particular: no encontrar ninguna clave que lo cubra → es la clave menor del fichero. Su lugar es el primer bloque.

Borrar un registro de datos con clave k

1. buscar el registro de datos con clave k para encontrar el bloque Bi dónde está.
2. modificar los bits lleno/vacío del bloque Bi. Se puede hacer compactación desplazando a la izquierda para no dejar huecos en medio.
3. si se borra el primer registro de un bloque hay que modificar el registro índice.
4. si el bloque se queda vacío → devolver el bloque al sistema. Encadenar los bloques anterior y siguiente. Eliminar la clave del índice, borrar el registro del índice encadenando los bloques.

4.8 Estructuras de árboles para índices de varios niveles B+

La estructura del fichero de índices puede ser simplemente secuencia de bloques encadenados.

Problema: cuando crecen los ficheros, el rendimiento de los índices secuenciales se degrada.. **Solución**:

4.8.1 Indices a varios niveles

Para mantener el acceso rápido: hacer un índice del fichero de índices.

Si todavía el fichero de índices sigue siendo grande: indexarlo a otor nivel más

El índice es un árbol cuyos nodos son bloques de índices al siguiente nivel

En los índices primarios, las hojas son los bloques del fichero de datos (no en los B+)

Se puede imponer que el árbol esté equilibrado en altura: todas las ramas tengan la misma longitud.

4.8.2 Índices de árboles equilibrados B+

Los árboles equilibrados B+ mantienen la eficiencia a pesar de las inserciones y borrados

Supone gasto extra de espacio y tiempo. Es aceptable en ficheros con frecuencia alta de actualizaciones.

B-tree(balanced tree) árbol equilibrado si los caminos de la raíz a los nodos hoja son de la misma longitud

La estructura de un nodo (raíz, interno u hoja) es:

P1	C1	P2	...	Pn-1	Cn-1	Pn

- Pi son punteros que apuntan
  - si el nodo es interno o raíz: a otro nodo de un nivel inferior
  - si es hoja: a un registro o a un cajón de punteros a registros
    (el cajón sólo se usa si el fichero no está ordenado según la clave de búsqueda)
- Ci son valores de la clave de búsqueda y están *ordenados*.
  - En el mismo nivel, los valores Ci no se repiten.
  - En el mismo nivel, los valores de uno nodo son todos anteriores (o posteriores) al de otro nodo.

Donde ***n*** es un valor decidido por el diseñador. En el ejemplo es n = 3.

Nodos internos (no hoja) tienen entre un mínimo $\lceil n/2 \rceil$ y máximo *n* hijos (o punteros) Los nodos internos forman un índice multinivel (disperso) sobre los nodos hoja (que apuntan a los registros de datos)

Asumiendo que un nodo puede tener ***m*** punteros, un puntero Pi, 1<i<m, apunta al nodo de nivel inferior que contiene los valores de la clave de búsqueda menores que Ci y mayores o **iguales** que Ci-1. El último puntero Pm apunta al nodo con valores de la clave mayores o **iguales** a Cm-1 (la clave anterior) y el primer puntero P1 apunta al nodo con valores de la clave menores que C1. Siempre hay un puntero más que claves. El último no tiene clave asociada.

Ejemplo:

P1 | 25 | P2 | 50 | P3

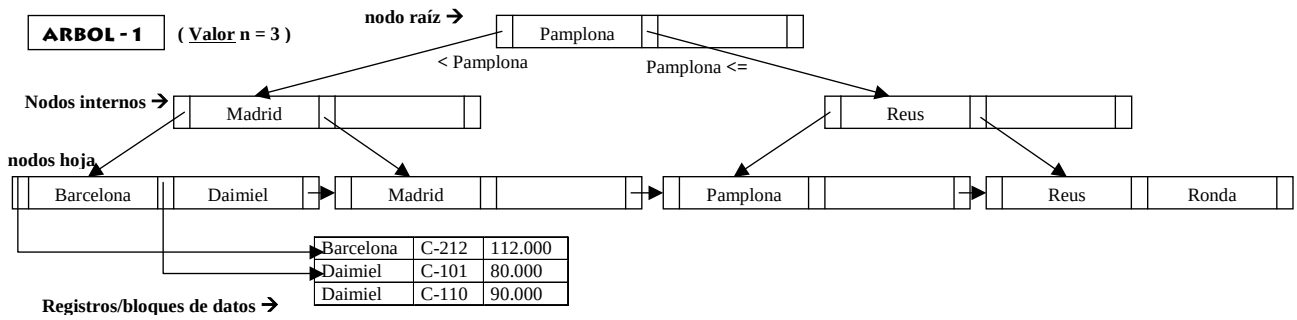
P1 apunta a la zona del árbol con valores menores que 25.

P3 apunta a la zona del árbol con valores mayores o iguales que 50.

P2 apunta a la zona del árbol con valores menores que 50 y mayores o iguales que 25.

Nodos hoja: contienen al menos (n -1) / 2 valores de claves (Ci) y máximo (n – 1) valores de Ci. El último puntero Pn de los nodos hoja encadena los *bloques* de nodos hoja. Esto permite un procesamiento secuencial eficiente del fichero. Los nodos forman un índice denso de las claves de registros de datos.

Nodo raíz: tiene entre 1 y *n* hijos o punteros (se permite menos del mínimo)



4.8.3 Consulta de un valor ***c***

Se examina el nodo raíz para buscar $\mathbf{c_i}$, que es el menor valor de la clave de búsqueda mayor que **c**. Seguimos el puntero pi asociado a la clave c_i hasta cierto nodo (sabemos que apunta a todos los menores). Si $c < c_1$ (la primera de todas las claves), entonces seguimos p2 hasta otro nodo. Si se tienen *m* punteros en el nodo y $c \geq c_{m-1}$ entonces se sigue p_m hasta otro nodo. Una vez más, se busca el menor valor de la clave de búsqueda que es mayor que *c* para seguir el puntero correspondiente. Finalmente se alcanza un nodo hoja, cuyo puntero apunta al registro o cajón deseado donde está **c**.

Camino en el árbol

Para procesar una consulta se tiene que recorrer un camino en el árbol desde la raíz hasta algún nodo hoja. Si hay *C* valores de la clave de búsqueda en el fichero, este camino no será más largo que $\lceil \log_{\lceil n/2 \rceil}(C) \rceil$ (que es la profundidad del árbol).

Ej: Bloque 4Kb, Clave 12 bytes, Puntero 8 bytes, caben **n**=200. Con clave de 32 bytes, **n**=100.

Con *n* = 100, 1.000.000 de valores de la clave -> $\lceil \log_{50}(1.000.000) \rceil = 4$ accesos a bloques. El nodo raíz se almacena en memoria intermedia -> 3 o menos accesos a disco.

Diferencia con los árboles binarios clásicos:

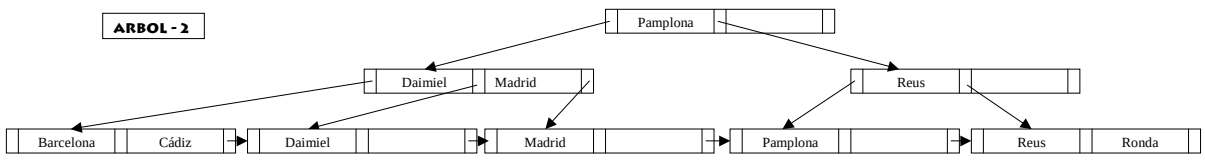
- Los nodos de los B+ son grandes y con muchos hijos, a diferencia de los pequeños nodos de los binarios con sólo dos hijos.
- Los B+ son anchos y bajos, y los binarios altos y estrechos.
- En binarios: $\lceil \log_2(1.000.000) \rceil = 20$ accesos a bloques frente a 4

4.8.4 Borrado e inserción

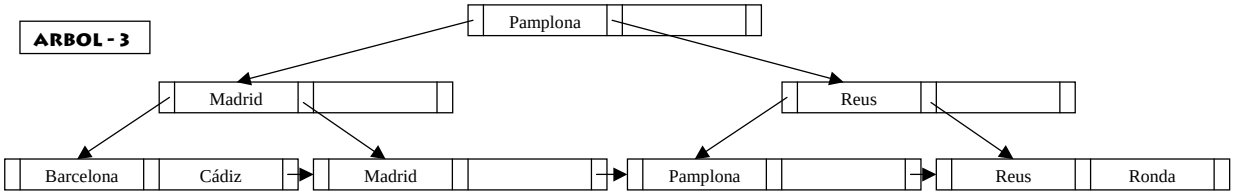
Borrado: Puede ser necesaria la recombinación de nodos si se viola $\lceil n/2 \rceil$.

Inserción: Puede ser necesaria la división de nodos si se viola n.

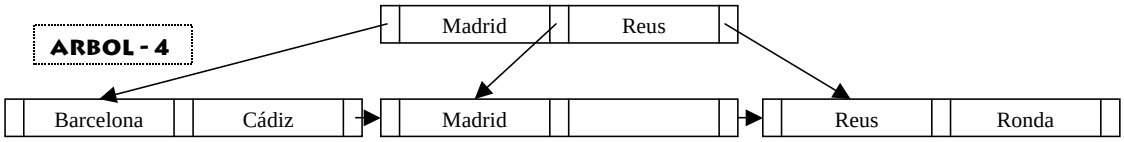
- **Inserción(sin esos problemas)**: se busca un nodo hoja donde tendría que aparecer el valor de la clave de búsqueda. Si el valor de la clave de búsqueda ya aparece en el nodo hoja, se inserta un nuevo registro en el fichero y, si es necesario, un puntero al cajón. Si el valor de la clave de búsqueda no aparece, se inserta el valor en el nodo hoja de tal manera que las claves de búsqueda permanezcan ordenadas. Luego insertamos el nuevo registro en el fichero y, si es necesario, creamos un nuevo cajón con el puntero apropiado
- **Borrado (sin esos problemas).** Usando la misma técnica que para buscar, se busca el registro a borrar y se elimina del fichero. Si no hay un cajón asociado con el valor de la clave de búsqueda o si el cajón se queda vacío como resultado del borrado, se borra el valor de la clave de búsqueda del nodo hoja
- **Inserción con división**: queremos insertar un registro en el árbol-1 anterior, cuyo valor de *nombre-sucursal* es “Cádiz” . Usando el algoritmo de búsqueda, “Cádiz” debería aparecer en el nodo que incluye “Barcelona” y “Daimiel”. No hay sitio para insertar el valor de la clave de búsqueda “Cádiz”. Por tanto, se *divide* el nodo en otros dos nodos. (árbol-2)



- **Borrado con recombinación**: queremos borrar "Daimiel" del árbol-2. Para ello se localiza la entrada "Daimiel" usando el algoritmo de búsqueda. Cuando se borra la entrada "Daimiel" de su nodo hoja, la hoja se queda vacía. Ya que en el ejemplo $n = 3$ y $0 < \lceil (n - 1)/2 \rceil$, este nodo se debe borrar del árbol B^+. Para borrar un nodo hoja se tiene que borrar el puntero que le llega desde su padre. En el ejemplo, este borrado deja al nodo padre, el cual contenía tres punteros, con sólo dos punteros. Ya que $2 >= \lceil n/2 \rceil$, el nodo es todavía lo suficientemente grande y la operación termina así (árbol-3).

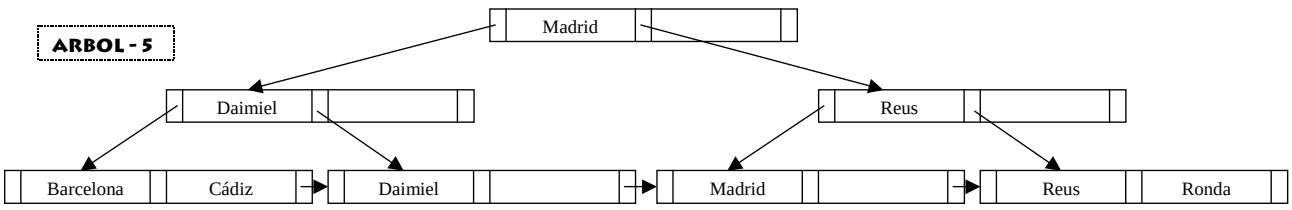


Cuando un borrado se hace sobre el padre de un nodo hoja, el propio nodo padre podría quedar demasiado pequeño. Si se borra "Pamplona" de la figura anterior (árbol-3), se provoca que un nodo hoja se quede vacío. Cuando se borra el puntero a este nodo en su padre, el padre sólo se queda con un puntero. Así, $n = 3$, $\lceil n/2 \rceil = 2$ y queda tan sólo un puntero. Sin embargo, ya que el nodo padre contiene información útil, no podemos simplemente borrarlo. En vez de eso, se busca el nodo hermano (el nodo interno que contiene al menos una clave de búsqueda, Madrid). Este nodo hermano tiene sitio para colocar la información contenida en el nodo que quedó demasiado pequeño, así que se funden estos nodos, de tal manera que el nodo hermano ahora contiene las claves "Madrid" y "Reus". El otro nodo (el nodo que contenía solamente la clave de búsqueda "Reus") ahora contiene información redundante y se puede borrar desde su padre. La raíz tiene solamente un puntero hijo como resultado del borrado, así que éste se borra y el hijo solitario se convierte en la nueva raíz. De esta manera la profundidad del árbol ha disminuido en una unidad.



No siempre es posible fundir nodos. Si borramos "Pamplona" del árbol-2. El nodo hoja que contiene "Pamplona" se queda vacío. El padre del nodo hoja se queda con sólo un puntero. De cualquier modo, en este ejemplo, el nodo hermano contiene ya el máximo número de punteros: tres. Así, no puede acomodar a un puntero adicional. La solución en este caso es *redistribuir* los punteros de tal manera que cada hermano tenga dos punteros. La redistribución de los valores necesita de un cambio en el valor de la clave de búsqueda en el padre de los dos hermanos.

Las operaciones de borrado e inserción son complicadas pero consumen poco tiempo



5. Implementación de las tablas por los SGBD

- *Cada tabla en un archivo separado*. Esto permite al SGBD aprovechar al máximo el sistema de archivos del SO. Normalmente, las tuplas de una relación pueden representarse como registros de longitud fija. Para sistemas pequeños.

- *El SGBD solicita al SO un archivo grande*, de forma que todas las tablas se guardarán en ese archivo y la gestión la realiza el propio SGBD sin usar el SO !!. Se puede optimizar la organización de registros en bloques por ejemplo haciendo agrupación en el mismo bloque de registros de distintas tablas (p.e. un registro de cliente y cada registro correspondiente a una cuenta de ese cliente, están en el mismo bloque o contiguos).