Contenido:

6. Organización física de los datos

6.1. Visión general de los sistemas de ficheros

6.1.1. Conceptos

- Campo:
  Def.: Elemento básico de datos.
  Contiene un valor único.
  Ejemplos: Nombre, Primer apellido, Segundo apellido
  Caracterizado por su tamaño y tipo. El tamaño puede ser fijo o variable (nada habitual).
- Registro:
  Def.: Colección de campos relacionados que se tratan como unidad en programas de aplicación.
  Ejemplo: Registro Persona, compuesto por los campos Nombre, Primer apellido, Segundo apellido
- Fichero:
  Def.: Colección de registros
  Entidad única para los programas y usuarios. Nombre único, pero pueden tener alias.
  Hay diferentes tipos de ficheros caracterizados por su organización de acceso.
  En sistemas dotados de seguridad, el acceso se puede limitar al fichero completo, a registros o a campos.
- Base de datos
  Colección de datos relacionados.
  Implementada con uno o varios ficheros del mismo o diferente tipo.
- Funciones

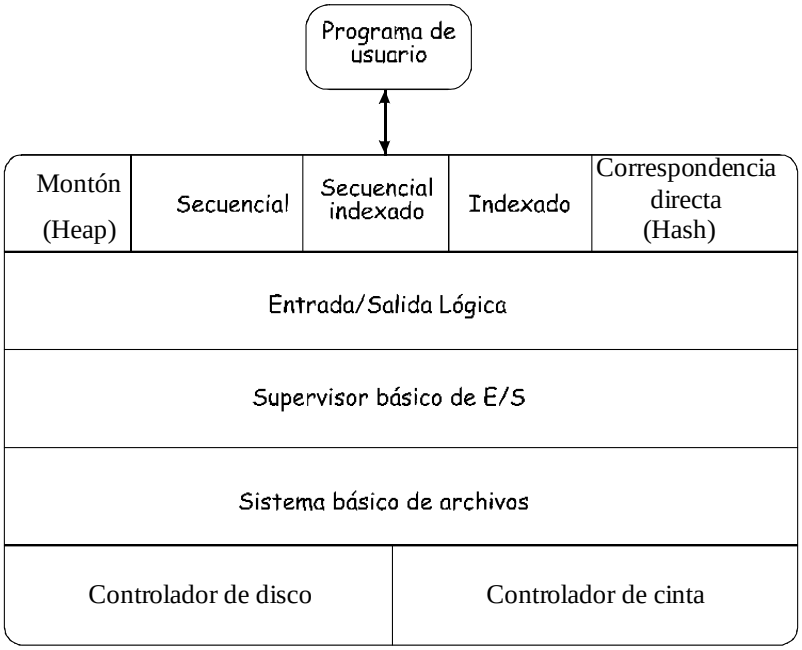
- Ficheros:
  Crear, Borrar, Cambiar nombre, Copiar, Trasladar (mover), Abrir, Cerrar
- Registros de un fichero:
  - Recuperar todo *. Recuperar todos los registros de un fichero. Equivalente a un acceso secuencial ya que los datos se recuperan uno a uno de esta forma.
  - Recuperar uno.
  - Recuperar siguiente.
  - Recuperar previo *.
  - Insertar uno *.
  - Borrar uno*.
  - Actualizar uno.
  - Recuperar varios.

6.1.2. Sistemas de gestión de ficheros

Sistema software que proporciona a los usuarios servicios relativos al uso de ficheros, evitando el diseño propio de software específico para el acceso a disco.
Servicios básicos:

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.

6.1.3. Arquitectura de los sistemas de ficheros



- Controladores (software) de dispositivos: Comunicación entre el controlador hardware del dispositivo y el sistema operativo.

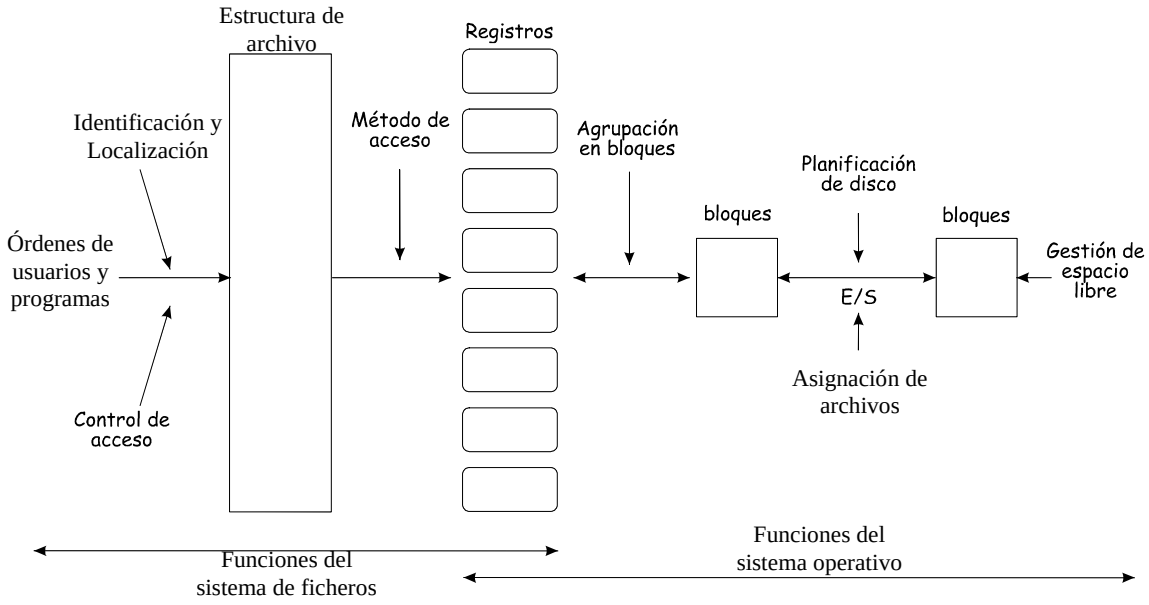
- Sistema básico de ficheros (Nivel físico de E/S): Trata con las transferencias de bloques de datos entre el almacenamiento principal y secundario.
- Supervisor básico de E/S: Inicio y terminación de las operaciones de E/S. Realiza la selección del dispositivo dependiendo del fichero requerido. Planifica 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.
- Método de acceso: Interfaz estándar entre las aplicaciones y el sistema de ficheros.

6.1.4. Funciones de la gestión de ficheros

Esquema básico de interacción con el sistema de ficheros:

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

La operación se solicita en términos de registros, pero ésta no es la organización del sistema de E/S.



6.1.5. Directorios

El directorio asociado a un conjunto de ficheros contiene información sobre los ficheros.
Es un fichero al que no pueden acceder usuarios ni programas de aplicación directamente (sólo a través de rutinas del sistema operativo).

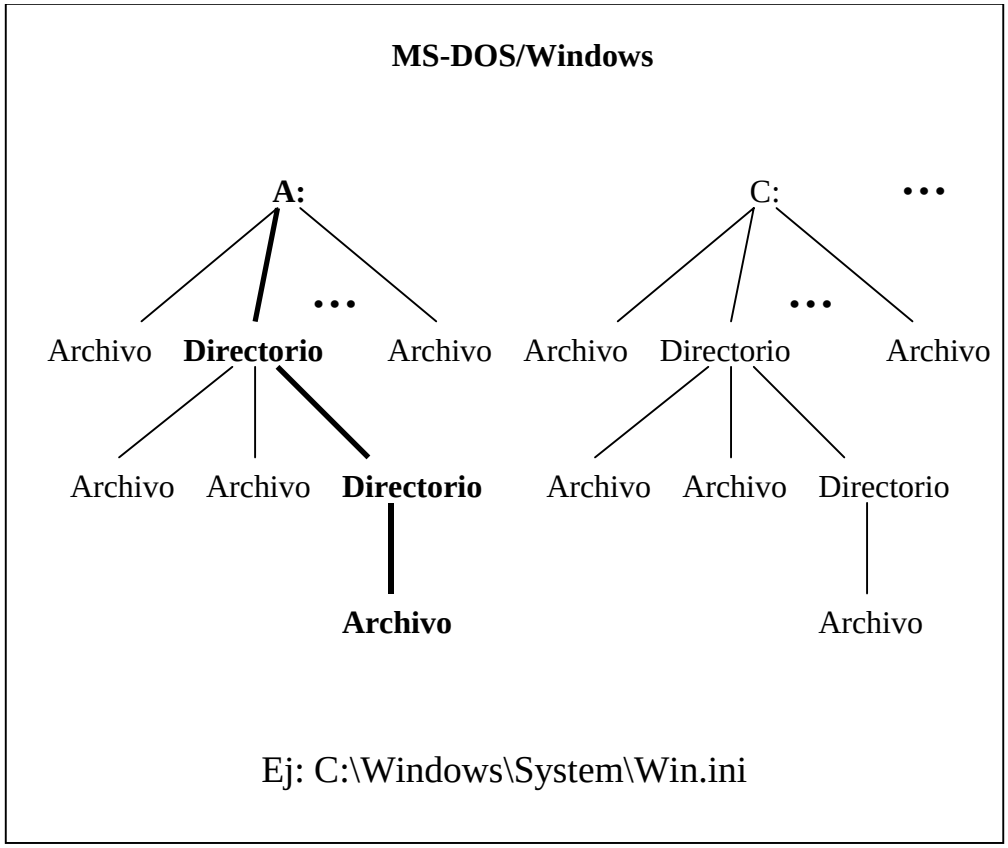
- Información básica
  - Nombre del fichero
  - Tipo de fichero
  - Organización del fichero
- Direccionamiento
  - Volumen
  - Dirección de inicio
  - Tamaño asignado
  - Tamaño utilizado
- Control de acceso
  - Propietario
  - Información de acceso
  - Acciones permitidas

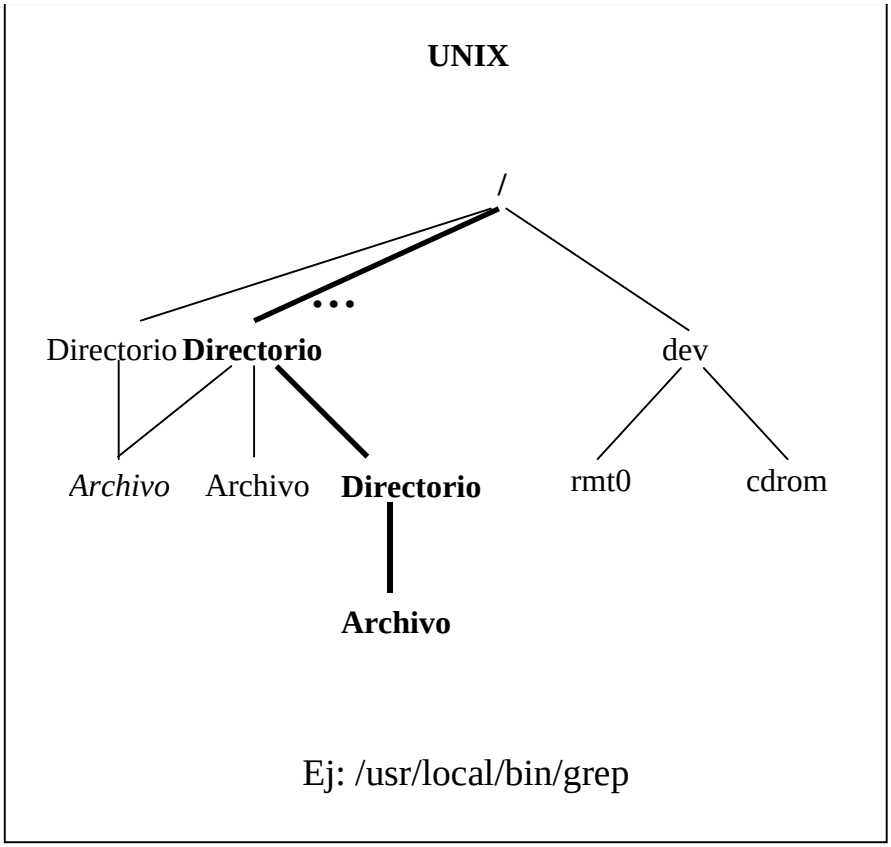
- Información de uso
  - Fecha de creación
  - Id Creador
  - Fecha última lectura
  - Id último lector
  - Fecha última modificación
  - Id Ultimo modificador
  - Utilización actual

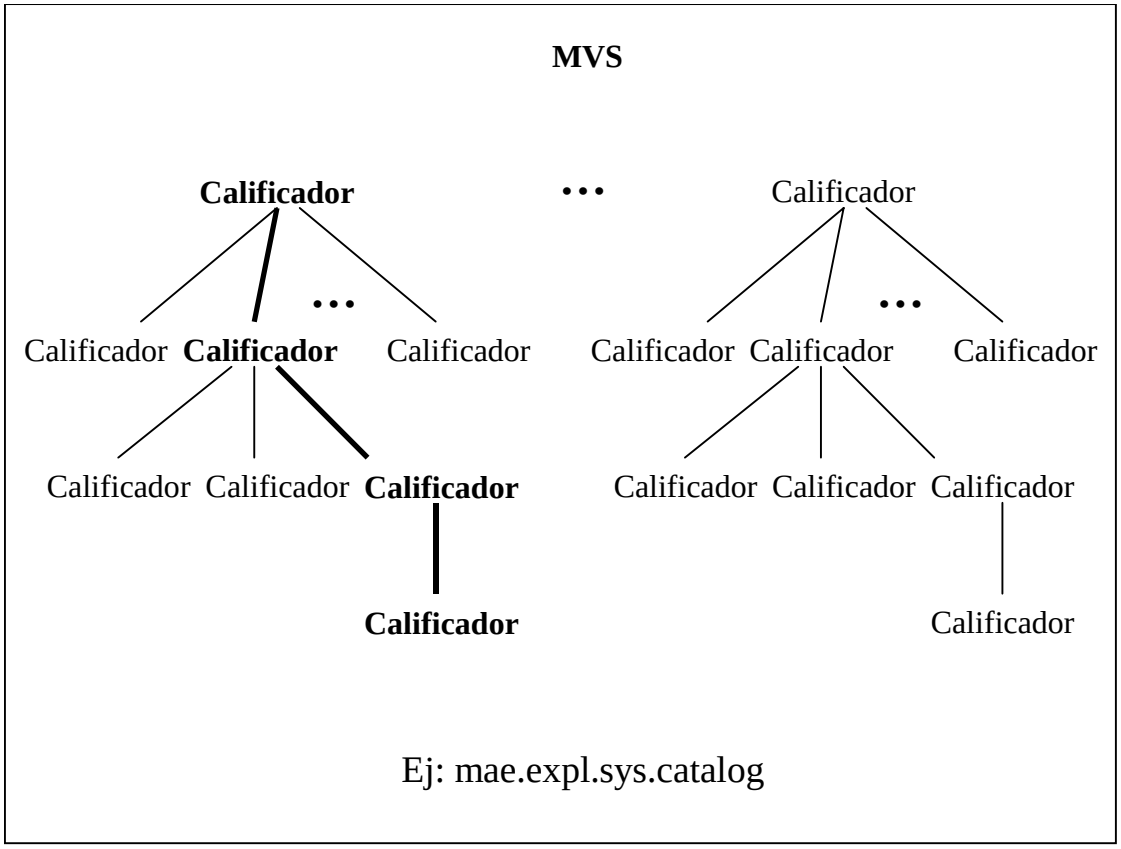
Designación de ficheros

Nombres únicos de ficheros
Clasificación de ficheros:
Directorios. Estructuras jerárquicas o grafos en cuyos nodos se clasifican los ficheros. Los nodos son ubicaciones lógicas de los ficheros en un sistema de clasificación.

- Ordenadores personales (MS-DOS,Windows). Estructura en árbol de directorios (o carpetas) y ficheros
- Estaciones de trabajo (UNIX). Estructura de red de directorios y ficheros
- Grandes sistemas centralizados (*mainframes*) . Estructura jerárquica de calificadores



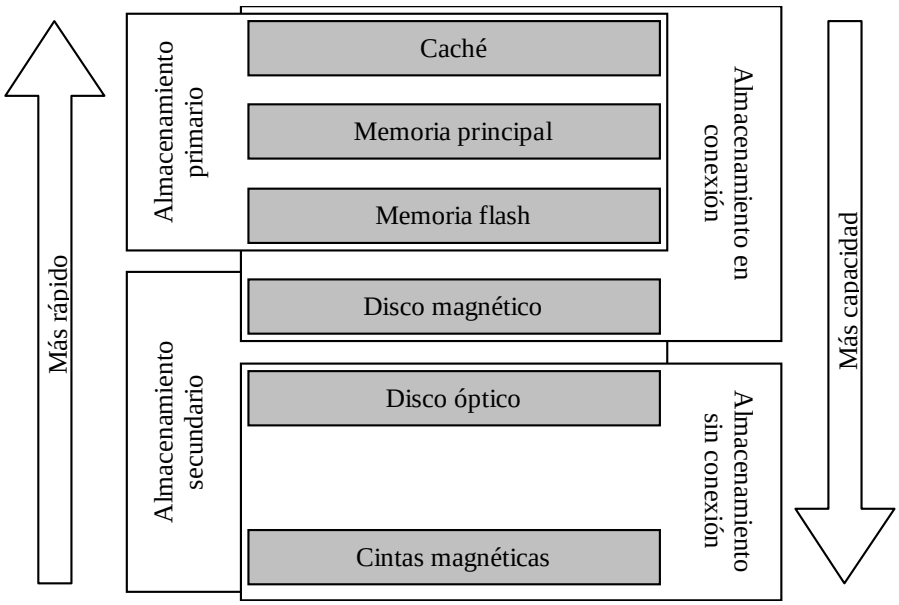




6.1.6. Medios de almacenamiento

Jerarquía (simplificada) de memoria:

- Almacenamiento primario: memoria volátil (caché y RAM)
- Almacenamiento secundario: discos magnéticos, memoria no volátil (*flash*)
- Almacenamiento terciario: discos ópticos y cintas



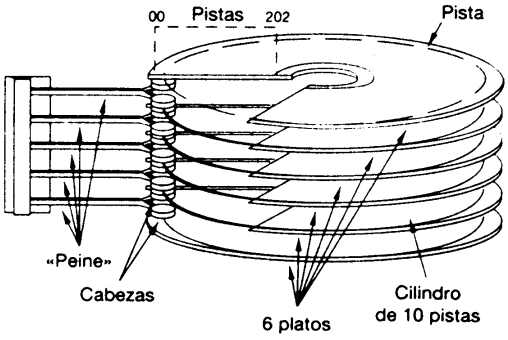
Memoria *flash*

- *Memoria de sólo de lectura programable y borrable eléctricamente* (*Electrically Erasable Programmable Read-Only Memory*, EEPROM).
- No volátil.
- Lectura: menos de cien nanosegundos, aproximadamente igual que la memoria principal.
- Escritura: Primera escritura, de cuatro a diez microsegundos. Sucesivas escrituras por bancos (se borra todo el banco y se puede volver a escribir).
- Inconveniente: número limitado de ciclos de borrado (diez mil - un millón).

Discos magnéticos

Acceso directo – secuencial

Características físicas de los discos



Un plato tiene dos superficies magnetizables. 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.

Los sectores varían desde los 32 hasta los 4096 bytes; generalmente son de 512 bytes.

500-1000 sectores por pista (internas-externas)

50.000-100.000 pistas en cada plato del disco

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.
La cabeza flota o vuela a sólo micras de la superficie del disco.
Un choque hace que el material magnético arrancado flote en el aire y se coloque entre las cabezas y sus platos, lo que causa más choques y provoca la destrucción del disco.
Planificación. Acceso a los bloques para minimizar el movimiento del brazo del disco: algoritmos de planificación del brazo del disco.
Organización de ficheros. Acceso secuencial: cilindros contiguos. Los sistemas operativos más antiguos (MVS de IBM para grandes sistemas) proporcionaban a los programadores un control detallado de la ubicación de los ficheros, lo que permitía a los programadores reservar un conjunto de cilindros para guardar un fichero. Actualmente no.
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 no volátil -> RAM alimentada por baterías)
Disco de registro histórico. Todos los accesos al disco de registro histórico son secuenciales, elimina el tiempo de búsqueda, pueden escribirse simultáneamente varios bloques consecutivos. También hay que escribir los datos en su ubicación verdadera, pero sin que el sistema de bases de datos tenga que esperar a que se complete.

Medidas del rendimiento de los discos

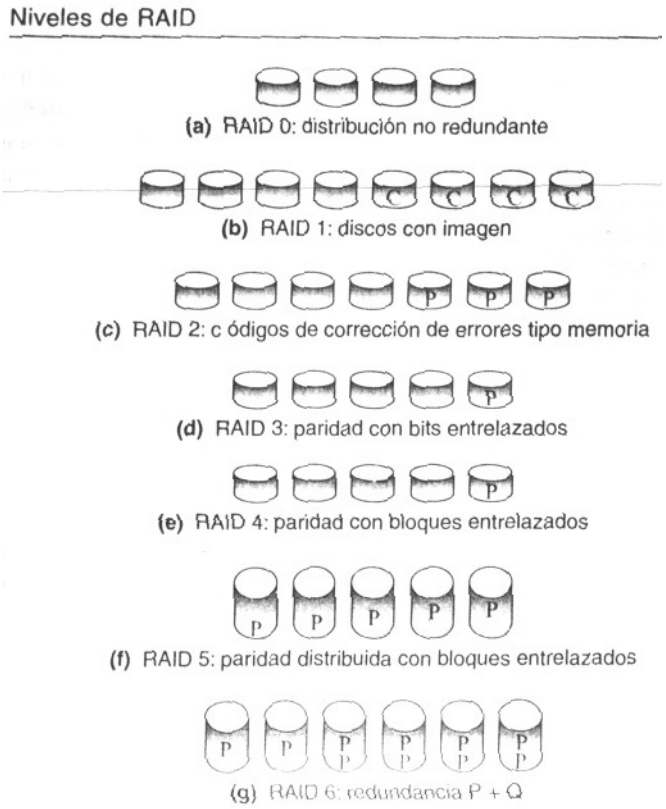
- *Tiempo de búsqueda*: es el tiempo necesario para volver a ubicar el brazo. Los tiempos de búsqueda típicos varían de 2 a 30 milisegundos. Los discos de menor tamaño tienden a tener tiempos de búsqueda menores. El tiempo medio de búsqueda es el promedio de los tiempos de búsqueda medido en una sucesión de solicitudes aleatorias (uniformemente distribuidas). El *tiempo medio de búsqueda* es la mitad del tiempo máximo de búsqueda. Los tiempos medios de búsqueda varían actualmente (año 2005) entre 4 y 10 milisegundos.
- *Tiempo de latencia rotacional*: es el tiempo de espera hasta que el sector al que hay que acceder aparece bajo la cabeza una vez que la cabeza haya alcanzado la pista deseada. Las velocidades rotacionales de los discos varían actualmente entre 5.400 rotaciones por minuto (90 rotaciones por segundo) y 15.000 rotaciones por minuto (250 rotaciones por segundo) o, lo que es lo mismo, de 4 a 11,1 milisegundos por rotación. El *tiempo medio de latencia rotacional* del disco es la mitad del tiempo empleado en una rotación completa del disco.
- *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. Primero hay que desplazar el brazo para que se ubique sobre la pista correcta y luego hay que esperar a que el sector aparezca bajo él debido a la rotación del disco. El tiempo de acceso es la suma del tiempo de búsqueda y del tiempo de latencia rotacional y varía de ocho a veinte milisegundos. Una vez situado bajo la cabeza el primer sector de los datos, comienza la transferencia.
- *Tiempo de transferencia de datos*: es el tiempo necesario para transferir datos una vez ubicada la cabeza.
- *Velocidad de transferencia de datos*: es la velocidad a la que se pueden recuperar datos del disco o guardarlos en él. Los sistemas de disco actuales soportan velocidades máximas de transferencia de veinticinco a cien megabytes por segundo; la velocidad de transferencia medias son significativamente menores que la velocidad de transferencia máxima para las pistas interiores del disco, dado que éstas tienen menos sectores (puede haber una diferencia de 3 a 1).

Ejemplo: para un disco de 7.200 rpm y 1.000 sectores por pista, ¿cuál es el tiempo de transferencia de un sector?
Frecuencia = 1/Periodo de rotación

t=1/7.200/60 $t = \frac{1}{7.200 rpm} \cdot \frac{1}{1000} = \frac{1}{7.200 \cdot \frac{1}{\min} \frac{1\min}{60 s}} \cdot \frac{1}{1000} = 8,33 \cdot 10^{-8} = 833 \mu s$

RAID

Actualmente, los discos pequeños son relativamente baratos.
Utilizar gran número de discos pequeños y baratos es más económico que utilizar un número menor que discos grandes y caros.
Rendimiento y fiabilidad: técnicas de organización de discos *disposición redundante de discos independientes* (*RAIDs - Redundant Array of Independent Disks*, antes *Redundant Array of Inexpensive Disks* – disposición redundante de discos de bajo coste).
Mejora de la fiabilidad: redundancia
Creación de imágenes o creación de sombras. Un disco lógico consiste en dos discos físicos y cada proceso de escritura se lleva a cabo en ambos discos.
Mejora del rendimiento: paralelismo
Imágenes de los discos: la velocidad de lectura se multiplica por el número de imágenes, dado que las solicitudes de lectura pueden enviarse a cualquiera de los discos
Distribución de datos: Distribución a nivel de bit. Con ocho discos se puede escribir el bit *i* de cada byte en el disco *i*.
Niveles de RAID
Redundancia a bajo costo combinando la distribución de discos con los bits de "paridad"
RAID 0: Distribución no redundante. Distribución de bloques sin redundancia. Se utiliza en las aplicaciones de alto rendimiento en las que la pérdida de datos no es crítica
RAID 1: Discos con imagen. RAID 0 + Imágenes. Almacenamiento de ficheros de registro histórico en sistemas de bases de datos, dado que ofrece el mejor rendimiento en procesos de escritura
RAID 2: Códigos de corrección de errores tipo memoria. Ej. distribución de bits de datos y bits de paridad o corrección de errores.
RAID 3: Paridad con bits entrelazados. Disco extra con información de paridad. Para grandes velocidades de transferencia de datos
RAID 4: Paridad con bloques entrelazados. Distribución de bloques + Disco de paridad
RAID 5: Paridad distribuida con bloques entrelazados. RAID 4 con distribución de bloques y paridad (aumenta también el rendimiento, al participar el disco extra en las transferencias). Se prefiere cuando las lecturas aleatorias son importantes, como ocurre en la mayor parte de los sistemas de bases de datos
RAID 6: Redundancia P+Q. RAID 5 con corrección de errores en lugar de sólo detección (Ej. Reed-Solomon). Pocas implementaciones, más fiable que RAID 5



Almacenamiento terciario

1. Almacenamiento óptico

Acceso secuencial/directo
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.
Tiempos de búsqueda: 250 ms
Tiempo de transferencia: 50Kb/s – 32x50Kb/s=1.600Kb/s
Capacidad: 500 Mb
DVD: 4,7-17Gb
Tipos de grabación, Informática [PLT97]

2. Almacenamiento en cinta

Acceso secuencial
Tiempo de acceso grande: minutos
Capacidad grande (5Gb)
Cinta de 8 mm (5Gb, 120 m), ¼", ½" (9 pistas)
Jukebox: Capacidad doce terabytes (10^{12} bytes)
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. Ej: Fichero sonoro digital de TVE.
Usos: Copias de seguridad, medio sin conexión para transferencia de información, almacenamiento de datos poco usados
Tipos de grabación: Longitudinal, transversal. Cintas de 7 y 9 pistas, DAT (4 mm), Video 8 mm [PLT97]

6.1.7. Memoria intermedia

Bloque: unidad de asignación de almacenamiento y de transferencia de datos

Un elemento de datos ocupa 1 bloque o menos (generalmente).

Gestor de la memoria intermedia

Los programas de aplicación solicitan bloques al gestor. Si el bloque está en memoria intermedia lo devuelve, si no, descarta algún otro bloque, que debe ser escrito en disco si ha sido modificado. Operación similar a la gestión de memoria virtual o memoria caché.
Estrategia de sustitución: generalmente LRU (menos recientemente utilizado, Least Recently Used).
Bloques clavados (pinned): se impide su salida. Importante para la recuperación de caídas.
Salida forzada de bloques: se fuerza su salida. Importante para la recuperación de caídas.

Estrategias de sustitución

El algoritmo
```
for each p in P do
  for each c in C do
    if p[nombre]=c[nombre] then
      incluir en el resultado un nuevo registro como combinación de p y c.
```
Suponiendo que las relaciones P y C se almacenan en dos ficheros diferentes.
Política de extracción inmediata: se saca el bloque que corresponda a p para el que se haya realizado el ciclo completo de C.
Para cada ciclo de C el usado menos recientemente es el que se necesitará más tarde: inverso de la política LRU. Es mejor usar MRU (más recientemente utilizado, Most Recently Used).
El gestor debe tener información estadística.
El diccionario de datos se tiene acceso frecuentemente: se deben clavar sus bloques y sólo desclavarlos cuando sea imprescindible.
También debe conservar los índices.
LRU, sorprendentemente, es la más utilizada a pesar de sus deficiencias.

- Subsistema de control de concurrencia
  Para conservar la consistencia puede posponer solicitudes de bloques, por lo que puede se modificar la estrategia de sustitución.
- Subsistema de control de caídas
  El gestor de la memoria intermedia actúa en coordinación con el subsistema de control de caídas para reescribir los bloques modificados en memoria.

6.2. Compartimiento de ficheros

Compartimiento de ficheros (simultáneo o no) entre distintos usuarios. Para que varios usuarios compartan ficheros es necesario definir derechos de acceso. Para que lo hagan simultáneamente, además hay que asegurar un acceso consistente.

6.2.1. Derechos de acceso

Los derechos básicos de acceso que pueden concederse a los usuarios son:

- Ninguno
- Conocimiento
- Ejecución
- Lectura
- Adición de información (escritura), pero sin modificar la existente
- Actualización: modificar, borrar y añadir datos del fichero
- Cambios de protección
- Eliminación

Estos derechos se pueden ver de forma jerárquica ya que cada uno engloba todos los anteriores.

Los derechos los asigna el propietarios (normalmente la persona que lo creó) en función del usuario que accede. Normalmente se distinguen tres tipos de usuario:

- Usuario específico. Los derechos se pueden asignar de forma individual a un usuario.
- Grupo de usuarios. Los derechos se pueden asignar de forma colectiva a un grupo de usuarios. El sistema operativo debe proporcionar soporte para los grupos.
- Todos los usuarios del sistema. Los derechos se pueden asignar de forma colectiva a todos los usuarios del sistema.

6.2.2. Accesos simultáneos

Disciplina para el acceso concurrente a los sistemas de ficheros.
Los problemas a abordar son los de la concurrencia: exclusión mutua e interbloqueo.
Las técnicas para su implementación pasan por los bloqueos en diferentes niveles:
- Fichero
- Registro
- Campo

6.3. Agrupación de registros

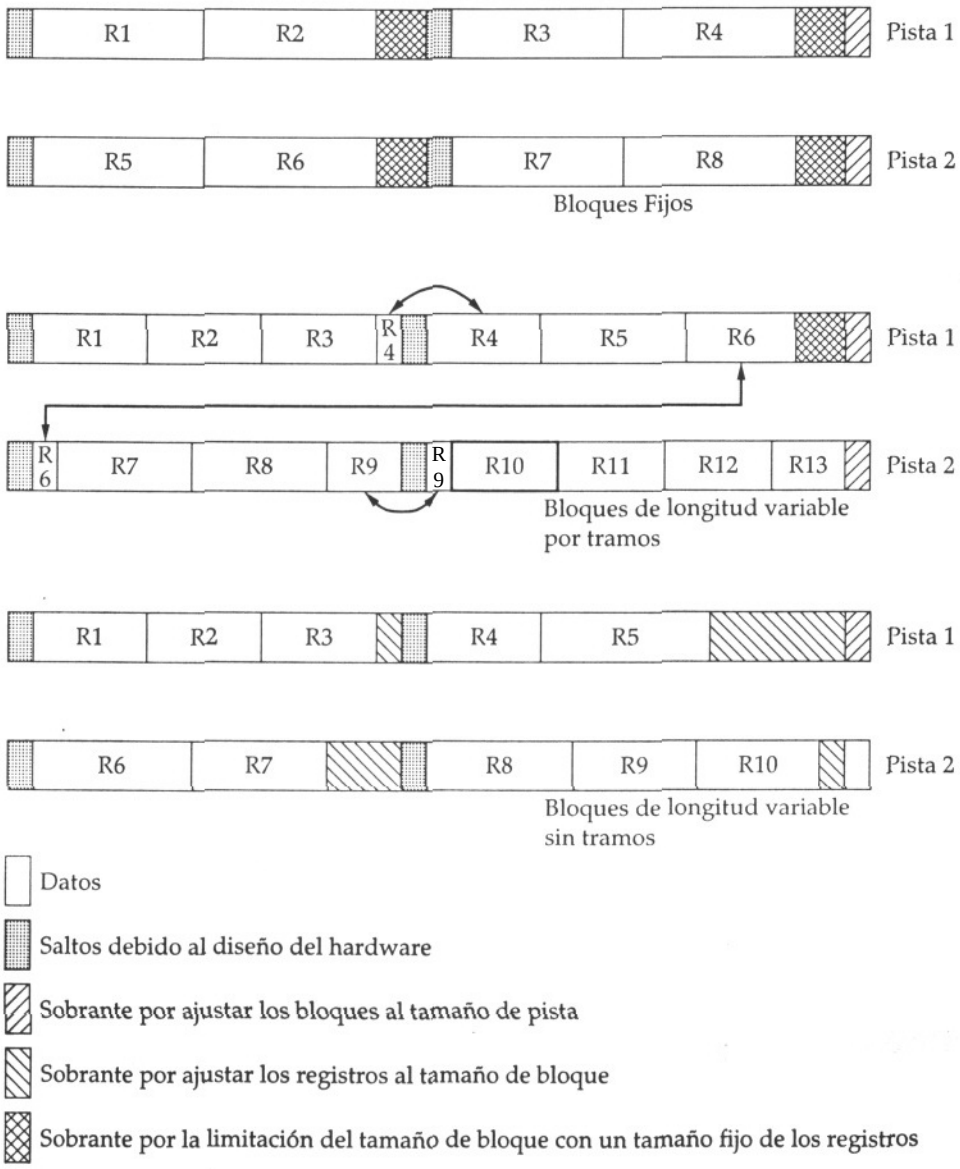
Registros (unidad lógica de acceso) vs. Bloques (unidad física de acceso, mínima unidad de transferencia entre memoria y el dispositivo periférico)
En MS/DOS y Windows, el bloque es una unidad de asignación (cluster).

6.3.1. Aspectos fundamentales de bloques

1. Los bloques pueden ser de tamaño variable aunque normalmente son de tamaño fijo para facilitar su gestión, en especial los búfer entre memoria principal y secundaria.
2. Tamaño relativo del bloque respecto al registro.
   1.1. En acceso secuencial, los bloques grandes disminuyen el tiempo de acceso al extraer varios registros en una única operación de E/S.
   1.2. En accesos directos, los bloques grandes desperdician la información contigua.
   Por lo general, bloques mayores reducen los tiempos de E/S

6.3.2. Agrupación en bloques

Factor de bloqueo: número de registros por bloque.
Tipos de agrupación:
1. Bloques fijos: Almacenan un número entero fijo de registros, pudiendo dejar parte del bloque sin utilizar. Método más frecuente.
2. Bloques de longitud variable por tramos: Registros de longitud variable que se agrupan en bloques, ocupando todo el espacio de los mismos. Algunos registros deben abarcar varios bloques, indicando con un puntero la conexión entre ellos. Economía de transferencias, pero difícil de implementar.
3. Bloques de longitud variable sin tramos: Registros de longitud variable pero sin división en tramos. En la mayoría de los bloques se desperdicia espacio. Desperdicio de espacio y limitación del tamaño de registro al de bloque.



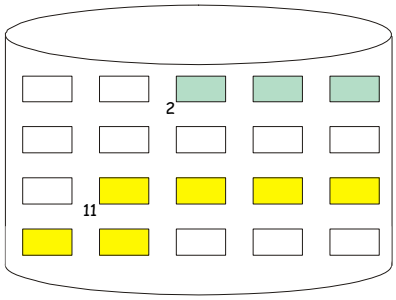
6.3.3. Asignación de memoria secundaria para ficheros

Tipos de asignación

Asignación previa: hay que predefinir un tamaño máximo de fichero en el momento de su creación.
Asignación dinámica: se asignan *secciones* (conjunto de bloques) dinámicamente. Es más elaborado y aparecen los problemas de fragmentación.

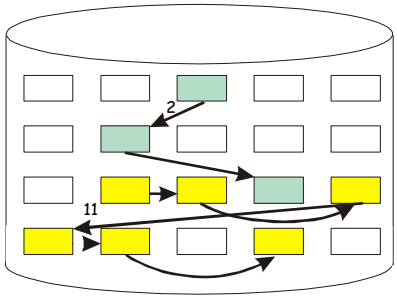
Métodos de asignación

- Asignación continua: Asignación de un conjunto continuo de bloques en el momento de la creación.
  Tabla de asignación de ficheros: sólo necesita una entrada por fichero indicando el comienzo y la longitud.
  Recuperación buena al traer varios bloques secuenciales o un único bloque *i* del fichero (Inicio + i).
  Desventaja: Fragmentación externa. Necesidad de compactación. Necesidad de la estimación previa del tamaño del fichero.



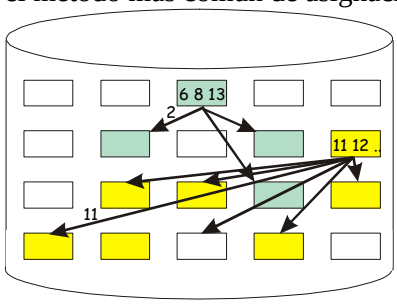
Nombre	Inicio	Tamaño
A	2	3
B	11	6

- Asignación encadenada: Asignación en bloques individuales que contienen un puntero al siguiente bloque.
  Tabla de asignación de ficheros: Una sola entrada por fichero con el inicio y el tamaño.
  Asignación: Cualquier bloque libre puede anexarse a la cadena.
  No existe fragmentación externa
  Problema: En la recuperación hay que recorrer el fichero entero para recuperar uno o varios bloques.



Nombre	Inicio	Tamaño
A	2	3
B	11	6

- Asignación indexada: Tabla de asignación con una entrada por fichero y con un índice a cada sección asignada del fichero.
  Tabla de asignación de ficheros: Los índices de la tabla suelen almacenarse en ficheros de índices asociados a cada fichero.
  Asignación: La asignación puede hacerse en bloques de tamaño fijo o en secciones de tamaño variable.
  Ventajas: Elimina la fragmentación externa. Soporta tanto el acceso secuencial como el acceso directo y por ello es el método más común de asignación.



Nombre	Indice
A	2
B	9

Gestión del espacio libre

Para la asignación de archivos es necesario conocer el espacio libre (bloques disponibles).
Tabla de asignación de disco: Indica el espacio libre al igual que la tabla de asignación de ficheros indica el ocupado.
Técnicas de implementación de la tabla de asignación de disco:

- Tablas de bits. Un vector con un bit por cada bloque del disco (0= bloque libre, 1= bloque usado). Poco espacio -> se puede conservar en memoria principal.
- Secciones libres encadenadas. Se necesita un puntero a la primera sección libre que a su vez apunta a la siguiente y así sucesivamente, y la longitud de la cadena
- Indexación. Trata el espacio libre como si se tratara de un archivo con la misma técnica de la asignación indexada de ficheros.

6.4. Organización de ficheros y métodos de acceso

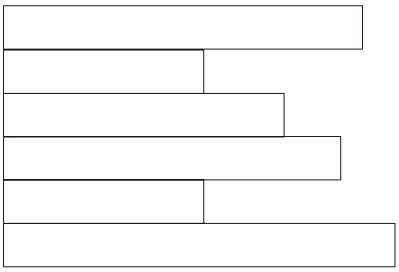
Fichero = Colección de registros
Organización de ficheros = Estructura lógica de los registros.
Criterios de organización:

- Acceso rápido
- Facilidad de actualización de la información
- Reducción de costes de almacenamiento
- Mantenimiento sencillo
- Fiabilidad para asegurar la integridad de los datos

La importancia de cada uno de estos factores depende del uso final al que esté destinado el sistema de ficheros.

6.4.1. Heaps (montones)

Organización menos estructurada.
Almacenamiento de los datos en el orden de llegada.
Cada campo debe tener un identificador y un tamaño asociado o un delimitador de final de registro.
Si se quiere recuperar un registro, hay que realizar una búsqueda exhaustiva en el heap.
Se aplica cuando se recogen y almacenan datos antes de procesarlos o cuando no es sencillo organizarlos.

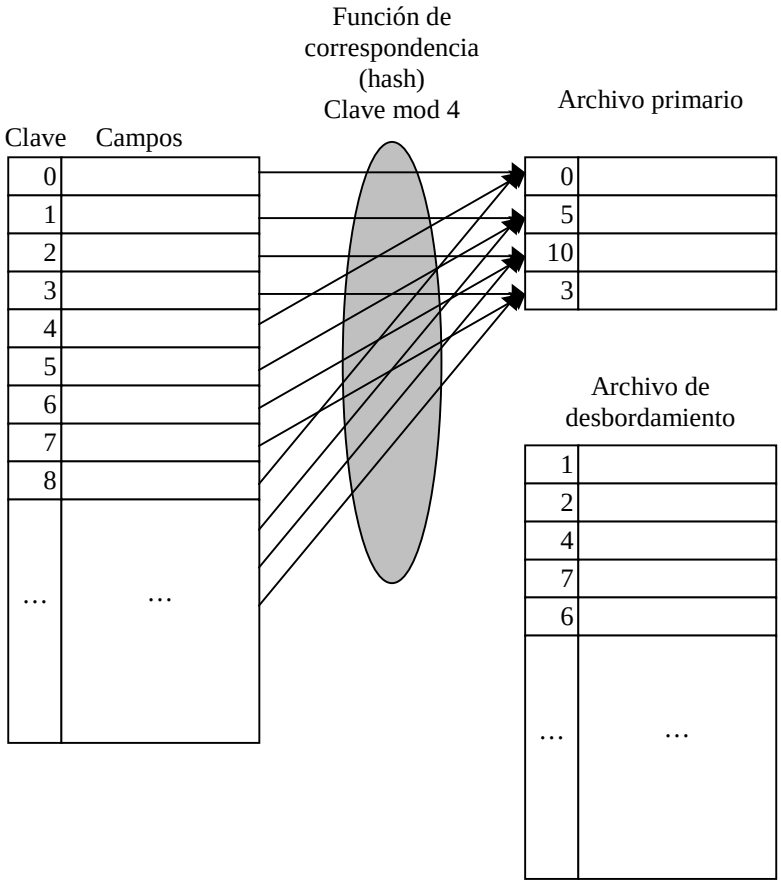


6.4.2. Ficheros secuenciales

Forma más común de organización de ficheros.
Se organizan registros de la misma longitud de formato fijo.
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.
Normalmente, la organización lógica se corresponde con la organización física almacenando conjuntos de registros en bloques de E/S que se almacenan secuencialmente en disco. Pero la inserción de nuevos registros es muy costosa. Por esta razón se usa un fichero de registro (log) en donde se almacenan los registros añadidos. Se utilizan punteros para localizarlos. Cada cierto tiempo se reorganiza el fichero para mantener la secuencia lógica de claves.

6.4.3. Ficheros con correspondencia directa (*hash*)

Aprovechan la capacidad de los discos para acceder a cualquier posición de dirección conocida. Se requiere un campo clave.



6.4.4. Ficheros con índices

Ficheros secuenciales indexados

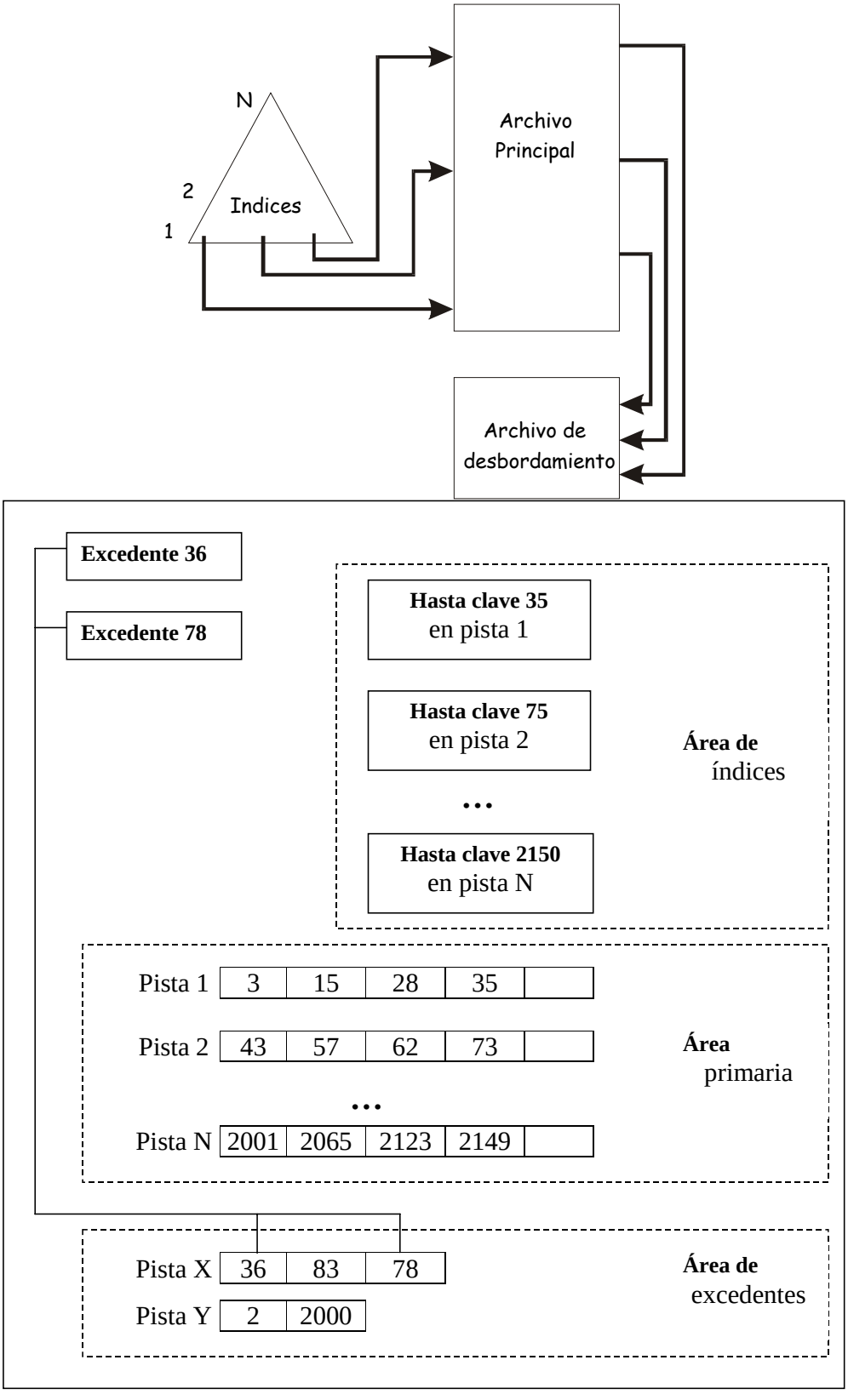
Los ficheros se organizan de forma secuencial pero se añaden dos características nuevas:

- un índice (o un conjunto de índices) para admitir accesos aleatorios
- un fichero de desbordamiento donde insertar los registros intermedios, junto con punteros a éstos, que parten de sus predecesores.

Esta organización reduce los tiempos de acceso sin sacrificar la esencia secuencial de la organización de bloques.

A mayor número de niveles de indexación, mayor complicación de gestión, pero mejor eficiencia en los acceso.

Limitación heredada de la organización secuencial: El procesamiento se limita al basado en un único campo clave del fichero.



Ficheros indexados

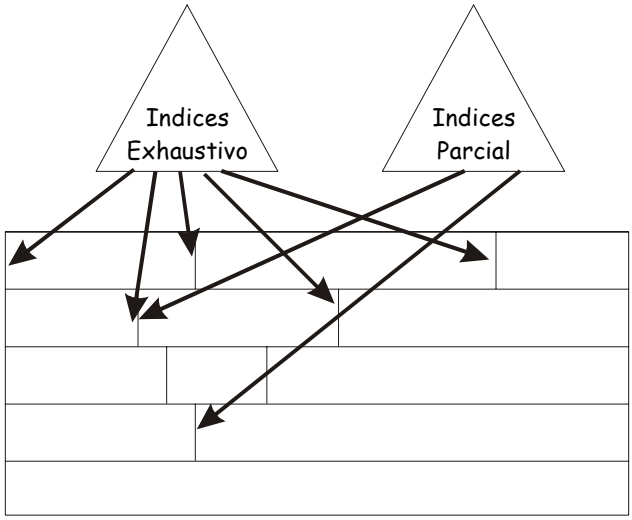
Estructura con índices múltiples, uno para cada campo que pueda ser objeto de búsqueda. El acceso a los registros se realiza sólo en función de sus índices.

Dos tipos de índices:

1. Indice exhaustivo con entradas a cada registro del fichero.

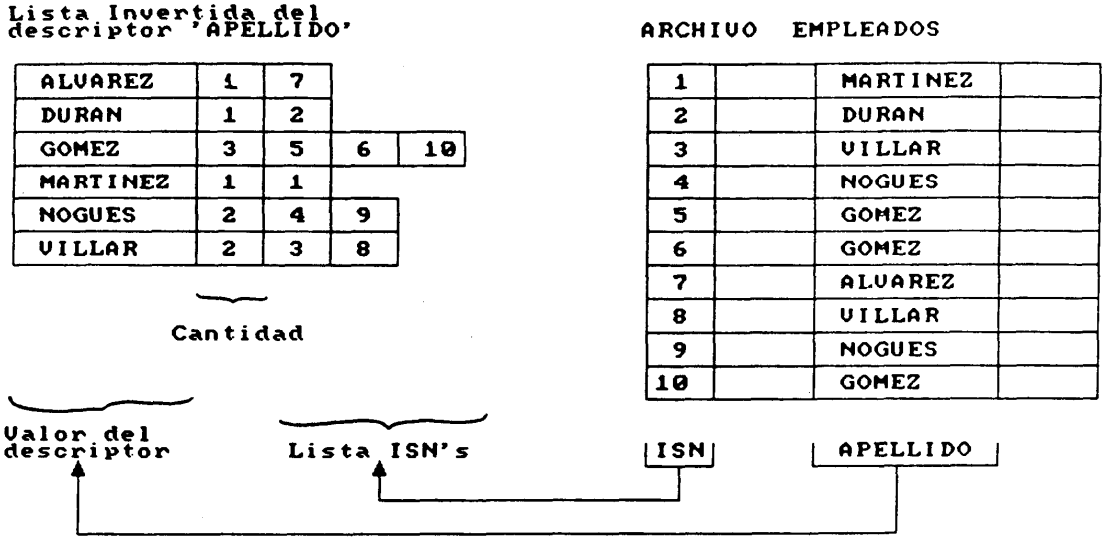
2. Índices parciales con entradas a los registros con un campo de interés (no todos los registros contienen todos los campos).

Cuando se añade un registro, hay que actualizar todos los índices.
Organización para aplicaciones donde la recuperación de registros es crítica y no realizan un cálculo exhaustivo sobre los registros.



Ficheros invertidos

Organizan el acceso con una clave de búsqueda, almacenando el número de registros con el valor de la clave, así como la ubicación de cada uno de ellos. Ejemplo:



Se usan en varios SGBD, por ejemplo, ADABAS ([EN00], Modelo de lista invertida (apéndice): [Dat93]).

Índices de mapas de bits

Índice de un fichero (una relación r) sobre un atributo A.
Mapa de bits: array de con tantos bits como registros tiene el fichero. Hay tantos mapas para el índice como valores distintos tenga A. Cada bit se corresponde con un registro del fichero: si es 1, el registro tiene el valor para el que se ha construido el mapa, si es 0, tiene un valor distinto.
Ej:

número de registro	*nombre*	*sexo*	*dirección*	*nivel-ingresos*
0	m	Juan	Pamplona	N1
1	f	Diana	Barcelona	N2
2	f	María	Jaén	N1
3	m	Pedro	Barcelona	N4
4	f	Cristina	Pamplona	N3

Mapas de bits para *sexo*:

m	10010
f	01101

Mapas de bits para *nivel-ingresos*

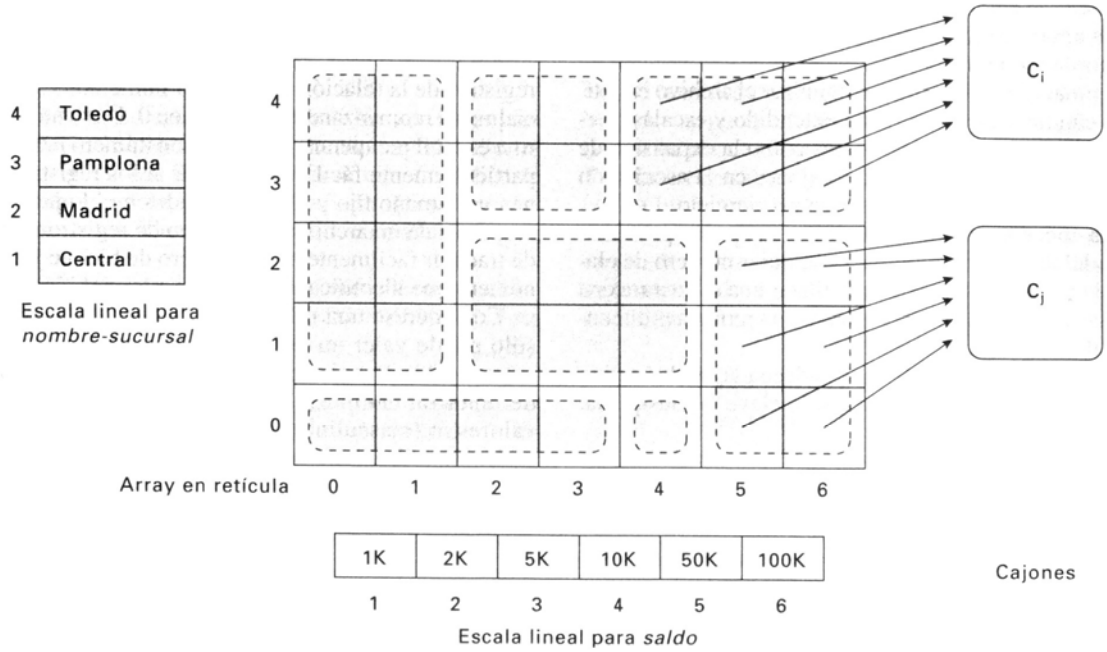
N1	10100
N2	01000
N3	00001
N4	00010
N5	00000

Mapas de bits de existencia:
Para marcar los registros reales y los borrados.

Utilidad de los mapas de bits:
1- Selecciones sobre varias claves cuando se espera un resultado pequeño.
Ej: Mujeres con ingresos de nivel N2 -> Intersección de los mapas de bits para *sexo* y *nivel-ingresos*
Selecciones conjuntivas: operación lógica and.
Selecciones disyuntivas: operación lógica or.
Negaciones: complemento (not). Problema con registros borrados y con valores nulos.
Implementación eficiente con el conjunto de instrucciones lógicas de bits del procesador.
2- Consultas sobre el número de registros (consultas de análisis de datos).
Ej: Número de personas que viven en Barcelona
Espacio ocupado: usualmente menor que el 1% del tamaño de la relación.

Ficheros en retícula

Compuestos de:
Arrays n-dimensionales: cada dimensión es un campo de búsqueda. Uno por cada fichero en retícula.
Escalas lineales: vectores con valores de cada campo de búsqueda. Son los vectores de valores de cada dimensión. Una por cada campo de búsqueda.
Cada celda del array contiene una dirección a un registro o a un cajón (conjunto de registros) con los valores indicados en las escalas lineales que corresponden a esa celda.
Las escalas lineales se deben escoger de forma que los registros estén uniformemente distribuidos en las celdas.
Al igual que con los mapas de bits, un único array puede responder a consultas de una o varias claves (hasta la dimensión del array).



6.4.5. Tipos de índices

Índices primarios

Se asume una ordenación sobre un campo clave (pueden aparecer varios registros con el mismo valor de la clave).

- Índice denso (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. 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).
- 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 (este índice denso tendría el papel del fichero secuencial). Si aún así el índice disperso es demasiado grande, se puede repetir el proceso tantas veces como sea necesario.

Índices secundarios

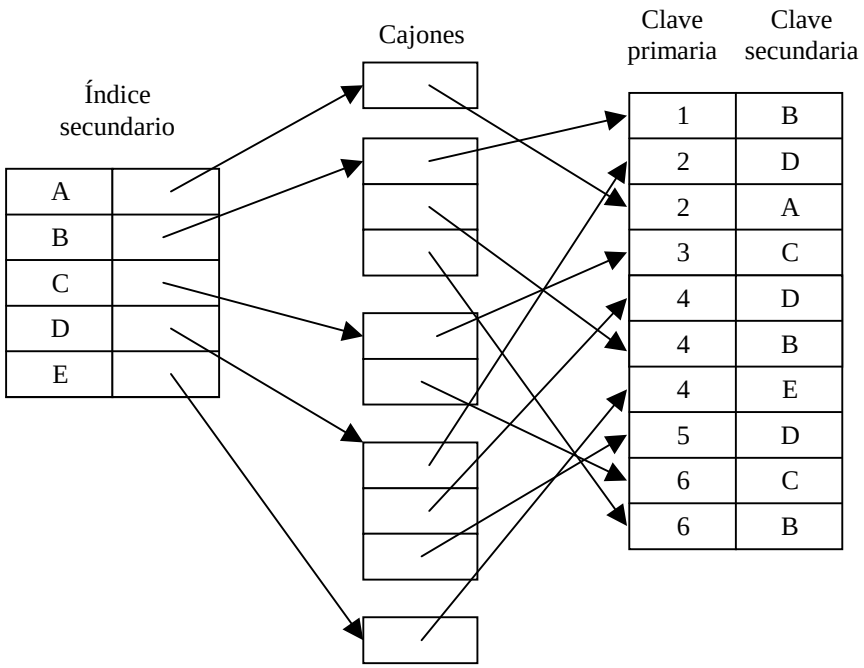
Aparecen cuando hay claves secundarias (candidatas), es decir, se accede a los registros por diferentes campos o cuando se quiere aumentar el rendimiento bajo determinados campos de búsqueda.

Como consecuencia, los sucesivos valores del índice no están almacenados secuencialmente.

Un índice secundario debe contener punteros a todos los registros, porque no se almacenan en la secuencia de claves.

Por la misma razón, deben ser índices densos (de otra manera no habría forma de acceder secuencialmente a partir de un valor que no esté en el índice).

Generalmente se implementa un segundo nivel de indirección: el primer nivel son todos los valores de la clave secundaria, y el segundo nivel son cajones (buckets) que almacenan todos los registros con ese valor (es la misma idea que las listas invertidas).



Índices de árboles equilibrados B+

Problema de los ficheros secuenciales indexados: degradación del rendimiento cuando crece el fichero.

Los árboles equilibrados B+ mantienen la eficiencia a pesar de las inserciones y borrados sin necesidad de la reorganización.

B-tree = balanced tree = árbol equilibrado = los caminos de la raíz a los nodos hoja son de la misma longitud.

Cada árbol B+ está caracterizado por un número *n* que se fija en su creación y depende de la cantidad de datos que pueda contener.

Cada nodo interno (no hoja) tiene entre $\lceil n/2 \rceil$ y *n* hijos, con *n* fijo para cada árbol particular.

El nodo raíz tiene entre 1 y *n* hijos.

Los nodos hoja contienen entre $\lceil (n-1)/2 \rceil$ y $n-1$ valores.

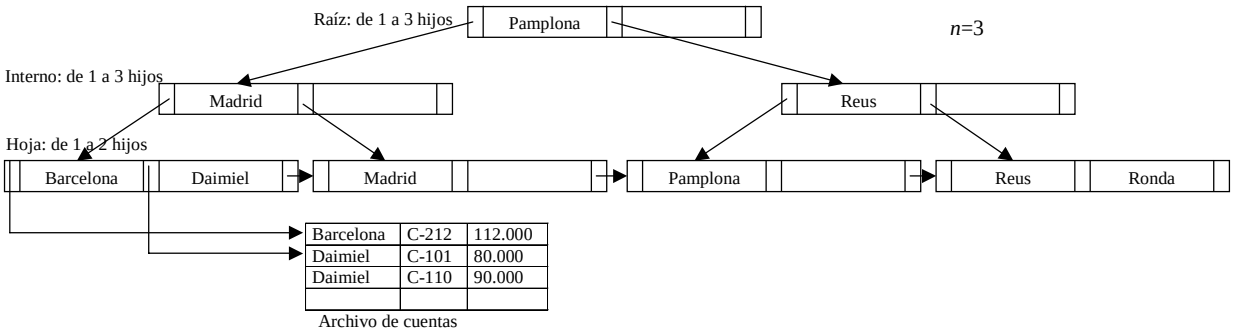
La estructura de un nodo (raíz, interno u hoja) es:

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

- Pi son punteros. Si el nodo es interno o raíz, el puntero apunta a otro nodo de un nivel inferior. Si es hoja, apunta a un registro o a un cajón de punteros a registros. El cajón sólo se usa si la clave de búsqueda no forma una clave primaria y si el fichero no está ordenado según la clave de búsqueda.
- Ci son valores de la clave y están ordenados. Los valores no se repiten. Para un mismo nivel del árbol los valores de un nodo son todos anteriores (o posteriores, según el orden) al de otro nodo.

El puntero Pi de un nodo raíz o interno apunta a un nodo con valores de la clave estrictamente menores que los de Ci, el puntero Pi+1 al nodo con valores mayores o iguales.

Los nodos internos del árbol B^+ forman un índice multinivel (disperso) sobre los nodos hoja.



Consulta de un valor c:
Se examina el nodo raíz para buscar el menor valor de la clave de búsqueda mayor que c. Supongamos que este valor de la clave de búsqueda es c_i . Seguimos el puntero c_i hasta cierto nodo. Si c < c_1 , entonces seguimos c_1 hasta otro nodo. Si se tienen *m* punteros en el nodo y c >= c_{m-1} entonces se sigue c_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.
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 $\lceil \log_{\lceil n/2 \rceil}(C) \rceil$.
Ej: Con bloque 4Kb, clave 12 bytes, puntero 8 bytes, n es del orden de 200. Con clave de 32 bytes, n=100.
Con *n* = 100 y 1.000.000 de valores de la clave -> $\lceil \log_{50}(1.000.000) \rceil = 4$ accesos a bloques.
Si el nodo raíz se almacena en memoria intermedia -> 3 o menos accesos a disco.

- Diferencia con los árboles binarios:
  - 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.
  - $\lceil \log_2(1.000.000) \rceil = 20$ accesos a bloques frente a 4

Borrado e inserción
Más difícil; para los nodos internos:
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.
Omitiendo estos problemas, las operaciones se realizarían:

- **Inserción**: 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.** 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 B^+ 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. El primero con $\lceil m/2 \rceil$ valores y el segundo con el resto. Se pueden provocar divisiones en los padres.

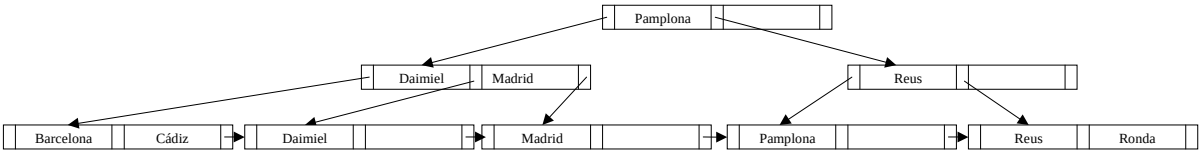
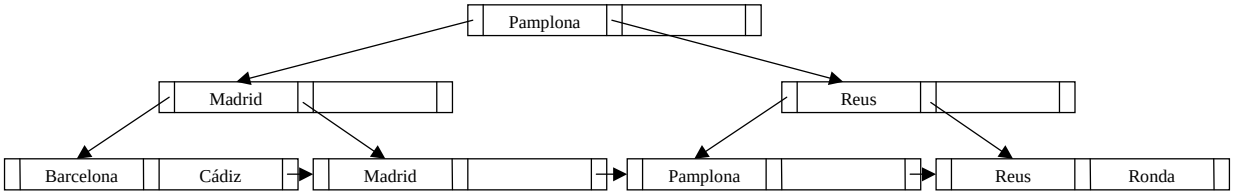
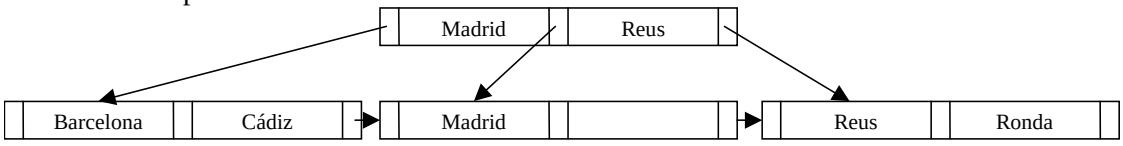


Figura 1

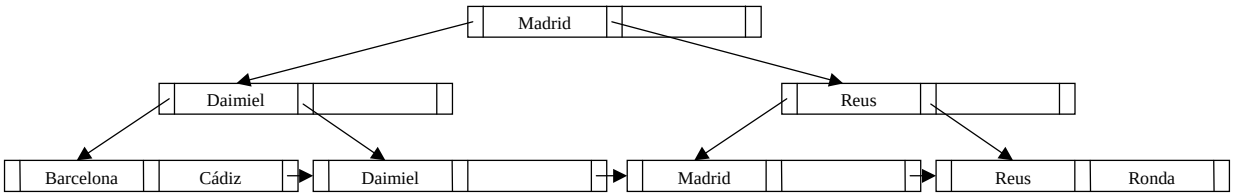
- **Borrado con recombinación**: queremos borrar “Daimiel” del árbol B^+ anterior. Para ello se localiza la entrada “Daimiel” usando el algoritmo de búsqueda. Cuando se borra la entrada “Daimiel” de su nodo hoja, la hoja se queda vacía. Ya que en el ejemplo $n = 3$ y $0 < \lceil (n - 1)/2 \rceil$, este 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 de borrado se completa.



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, 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.



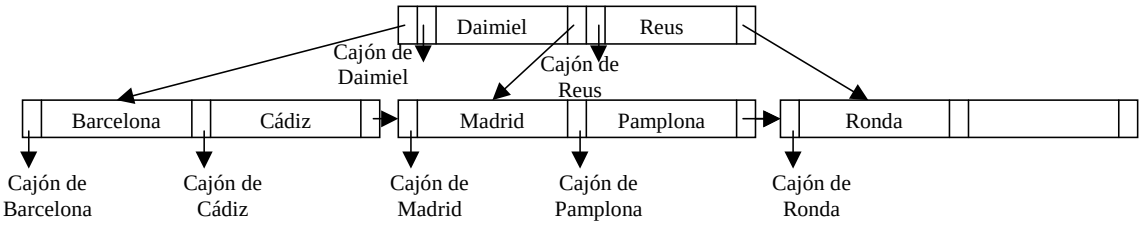
No siempre es posible fundir nodos. Como ejemplo se borrará “Pamplona” del árbol B^+ de la Figura 1. El nodo hoja que contiene “Pamplona” se queda vacío. El padre del nodo hoja se queda con sólo un puntero. De cualquier modo, en este ejemplo, el nodo hermano contiene ya el máximo número de punteros: 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

Índices de árboles equilibrados B

Similares a los B+, pero eliminan los valores redundantes. Ej. equivalente a la Figura 1:



Dado que las claves de búsqueda que aparecen en los nodos internos no aparecen en ninguna otra parte del árbol B, es necesario un campo adicional para un puntero por cada clave de búsqueda de un nodo interno. Estos punteros adicionales apuntan a registros del fichero o a los cajones de la clave de búsqueda asociada.

Nodo hoja:

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

Nodo interno:

P1	B1	C1	P2	B2	C2	...	Pm-1	Bm-1	Cm-1	Pm

Hay n - 1 claves en el nodo hoja, mientras que hay m - 1 claves en el nodo interno. Ocurre porque los nodos internos deben incluir los punteros B_i, reduciendo el número de claves de búsqueda que pueden contener estos nodos.

$m < n$, pero la relación exacta entre m y n depende del tamaño relativo de las claves de búsqueda y de los punteros.

Cada nodo, excepto el raíz y los hoja, tiene al menos $\lceil n/2 \rceil$ hijos.

En algunos casos, las búsquedas tienen éxito antes de recorrer todo el árbol en profundidad.

Al tener mayor anchura que profundidad, la probabilidad de encontrar valores que no estén en nodos hoja es pequeña.

Los nodos del árbol B tienen menor grado de salida que los del B+, por lo que será en general más profundo.

No obstante, el tiempo de búsqueda sigue siendo proporcional al logaritmo del número de valores de la clave de búsqueda.

El borrado y la inserción son más complicados: el registro puede estar en un nodo interno.

Inserción y borrado

- **Inserción**
  Localizar el nodo hoja en el que el nuevo par clave-puntero se debería insertar.
  Si hay espacio sobrante en el nodo hoja, realizar la inserción en la ubicación correcta y la tarea se completa.
  De lo contrario, insertar el par clave-puntero conceptualmente en la ubicación correcta del nodo hoja y partirlo por la mitad.
  El par intermedio clave-puntero no va dentro de los nodos resultantes de la operación dividir. En su lugar, se inserta en el nodo padre junto al puntero del árbol para el nuevo hijo. Si no hay espacio en el padre, se repite el proceso.
- **Borrado**
  Localizar el valor el valor de la clave a borrar, en el árbol B.

a) Si se encuentra un nodo hoja, borrar el par clave-puntero y el registro del fichero. Si el nodo hoja contiene menos de $\lceil n/2 \rceil - 1$ entradas como resultado de este borrado, o bien se fusiona con sus hermanos o algunas entradas se redistribuyen para ello.
La fusión implicaría un borrado, y la redistribución cambios en las entradas del nodo del padre. Los borrados pueden llegar hasta la raíz del árbol B.

b) Si el valor de la clave se encuentra en un nodo interno del árbol B, reemplazar este valor y su puntero de registro por el menor valor de la clave en el subárbol inmediatamente a su derecha y el correspondiente puntero del registro. Borrar el registro actual en el fichero de la base de datos. Borrar después el menor valor del par clave-puntero del subárbol. Este borrado pueden originar borrados hasta la raíz del árbol B.

Asociación estática (static hashing)

Evita el uso de índices: menos accesos de disco.
Se usa una función del valor 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 puede asignarse a varios valores de la clave).
Generalmente un cajón es un bloque de disco.
En un cajón puede haber registros con valores de clave diferente. Para completar una búsqueda hay que examinar en general todos los registros de un cajón.
La inserción y la búsqueda son, pues, operaciones sencillas.

Consideraciones sobre la función de asociación:

- Distribución uniforme. Cada cajón tiene asignado el mismo número de valores de entre todos los posibles. No significa que el almacenamiento sea uniforme, a menos que la probabilidad de aparición de los valores sea igual.
- Distribución aleatoria. Asignación aleatoria. En media, la ocupación será uniforme.

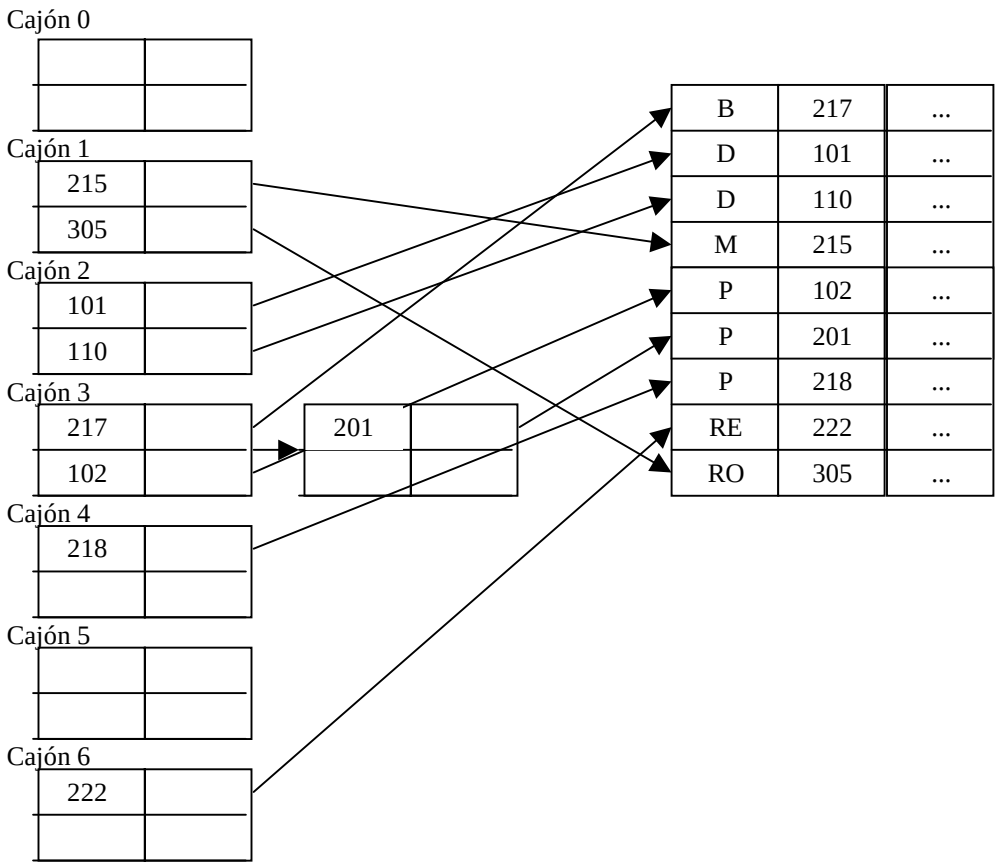
Gestión de cajones:

- Cajones Suficientes. Se debe cumplir: número de cajones > número de registros/capacidad cajón. Generalmente se escoge número de cajones = número de registros/capacidad cajón*(1+d), siendo d=0,2. Se pierde alrededor del 20% de espacio.
- Atasco de cajones. Algún cajón se puede llenar aunque haya espacio en otros

Cuando se produce atasco de cajones se usan cajones de desbordamiento enlazados, se denomina cadena de desbordamiento.

Índices asociativos

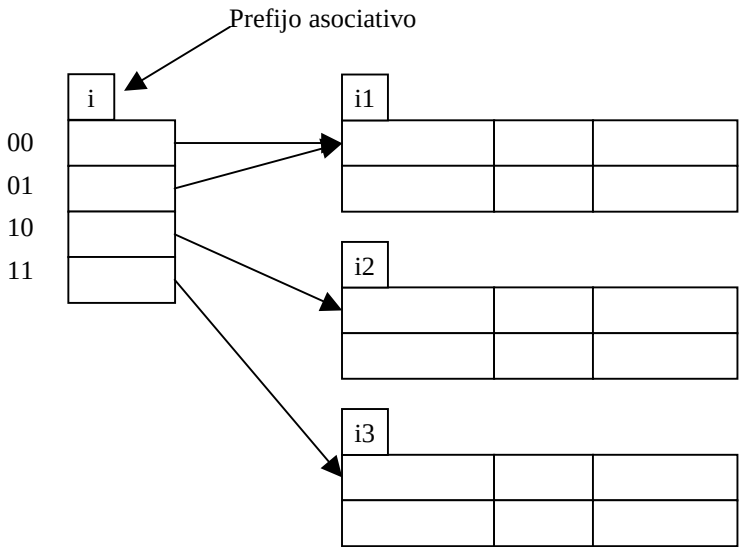
Se pueden usar índices asociativos con cajones. Ej:



Asociación dinámica

El problema de la asociación estática es el crecimiento de los ficheros.
Si se elige una función de asociación para el tamaño actual del fichero se degradará el rendimiento cuando la BD crezca.
Posible solución: Reorganización; pero requiere mucho tiempo y se impide el acceso a la BD.
Con la asociación dinámica se permite modificar la función de asociación dinámicamente.
Una forma de asociación dinámica es la asociación extensible.
La asociación extensible divide y recombina los cajones a medida que la BD aumenta y disminuye respectivamente. Por tanto, la reorganización se hace por cajones, en lugar de por toda la BD.
Generalmente la función de asociación extensible puede generar 2^{32} valores diferentes (más de 4 mil millones). No se crea un cajón por valor, sino bajo demanda.
Inicialmente se escogen i bits <= 32, y este valor aumenta o disminuye según el crecimiento o decrecimiento de la BD.
En un momento determinado se crea un cajón para un determinado i, y se anota para ese cajón el número i.
El número de entradas en la tabla de direcciones que apuntan al cajón j es $2^{(i-ij)}$.
Prefijo asociativo
i
00
01
10
11
Tabla de direcciones

i1 Cajón 1
i2 Cajón 2
i3 Cajón 3



La localización de un cajón se realiza a través de la tabla de direcciones.
La inserción sigue el mismo procedimiento y se llega a un cajón j. Si tiene sitio, se inserta; si no, hay que dividir el cajón.

- Si i=ij sólo una entrada en la tabla de direcciones apunta al cajón j. Es necesario incrementar el tamaño de la tabla de direcciones. Se hace i=i+1 (se duplica el espacio para los cajones). Cada entrada de la tabla se sustituye por dos entradas, cada una con el mismo puntero que la entrada original. Ahora dos entradas en la tabla de direcciones de cajones apuntan al cajón *j*. Así pues se asigna un nuevo cajón (cajón *z*) y hacemos que la segunda entrada apunte al nuevo cajón. Se pone i_j e i_z a *i*. A continuación se vuelve a calcular la función de asociación para cada registro del cajón *j* y, dependiendo de los primeros *i* bits (recuérdese que se ha añadido uno a *i*), se mantiene en el cajón *j* o se coloca en el cajón recién creado.
  Se vuelve a intentar la inserción del nuevo registro. Normalmente el intento tiene éxito. Si todos los registros del cajón *j*, así como el nuevo registro, tienen el mismo prefijo del valor de la función de asociación, será necesario dividir el cajón de nuevo, ya que todos los registros en el cajón *j* y el nuevo registro tienen asignados el mismo cajón. Si la función de asociación se eligió cuidadosamente, es poco probable que una simple inserción provoque que un cajón se divida más de una vez, a menos que haya un gran número de registros con la misma clave de búsqueda. Si todos los registros en el cajón *j* tienen el mismo valor de la clave de búsqueda, ningún número de divisiones servirá. En estos casos se usan cajones de desbordamiento para almacenar los registros, como en la asociación estática.

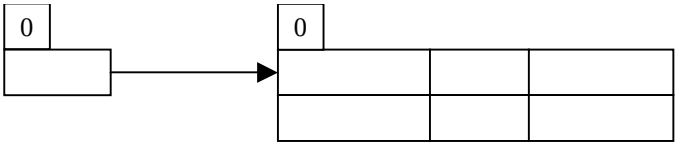
  Ejemplo:

Barcelona	C-217	150.000
Daimiel	C-101	100.000
Daimiel	C-110	120.000
Madrid	C-215	140.000
Pamplona	C-102	80.000
Pamplona	C-201	180.000
Pamplona	C-218	140.000
Reus	C-222	140.000
Ronda	C-305	70.000

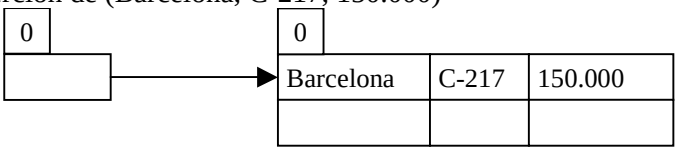
nombre-sucursal	*h*(*nombre-sucursal*)
Barcelona	0010 ...
Daimiel	1010 ...
Madrid	1100 ...
Pamplona	1111 ...
Reus	0011 ...

Ronda 1101 ...

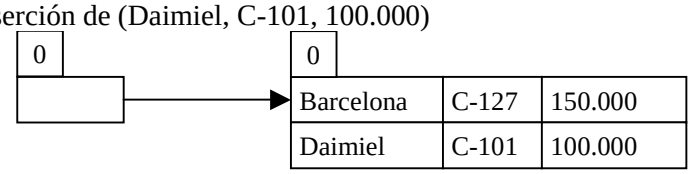
Un cajón: 2 registros
Originalmente fichero vacío.



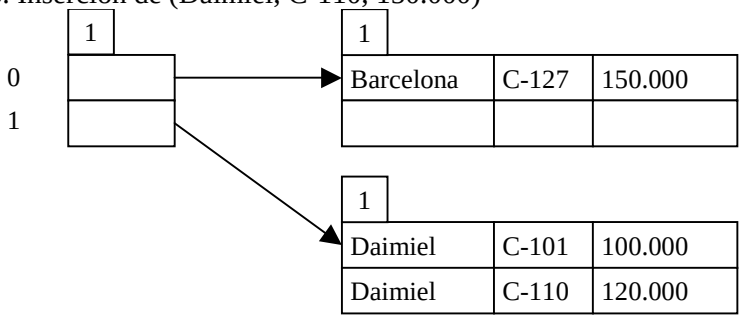
1. Inserción de (Barcelona, C-217, 150.000)



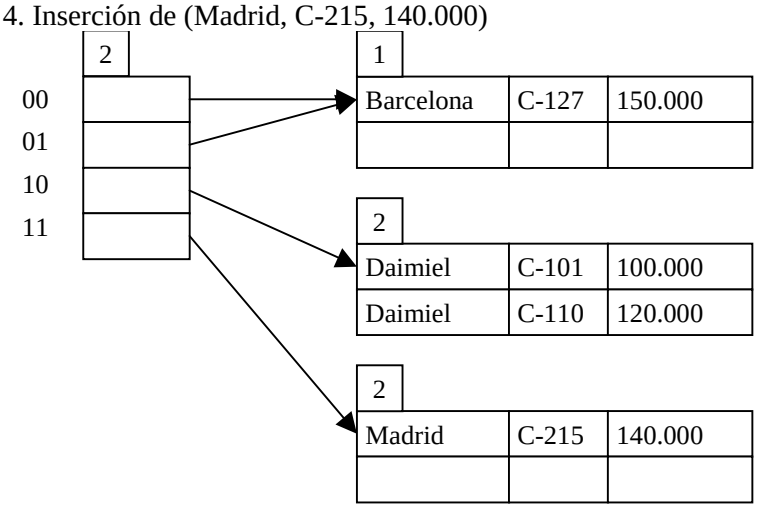
2. Inserción de (Daimiel, C-101, 100.000)



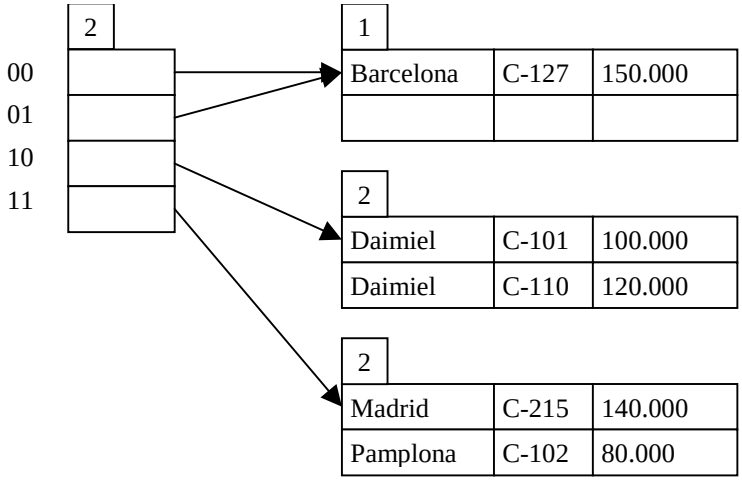
3. Inserción de (Daimiel, C-110, 150.000)



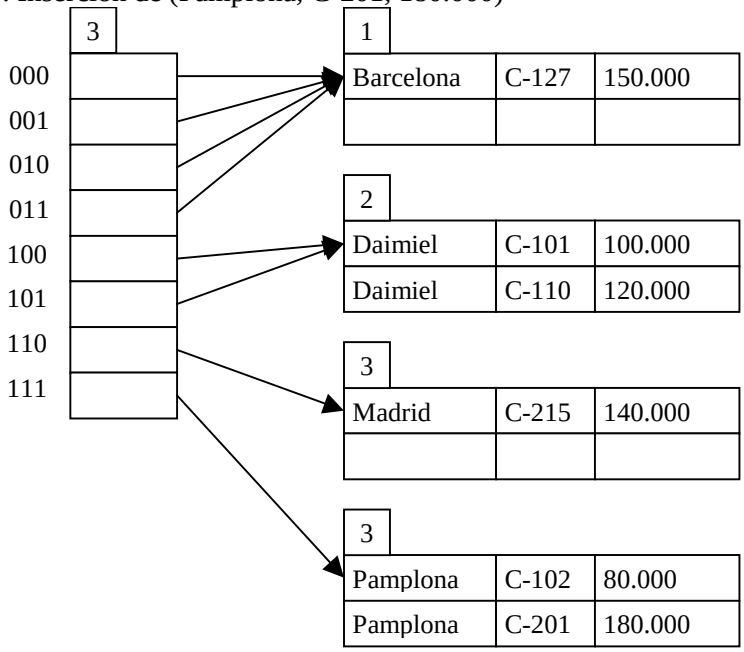
4. Inserción de (Madrid, C-215, 140.000)



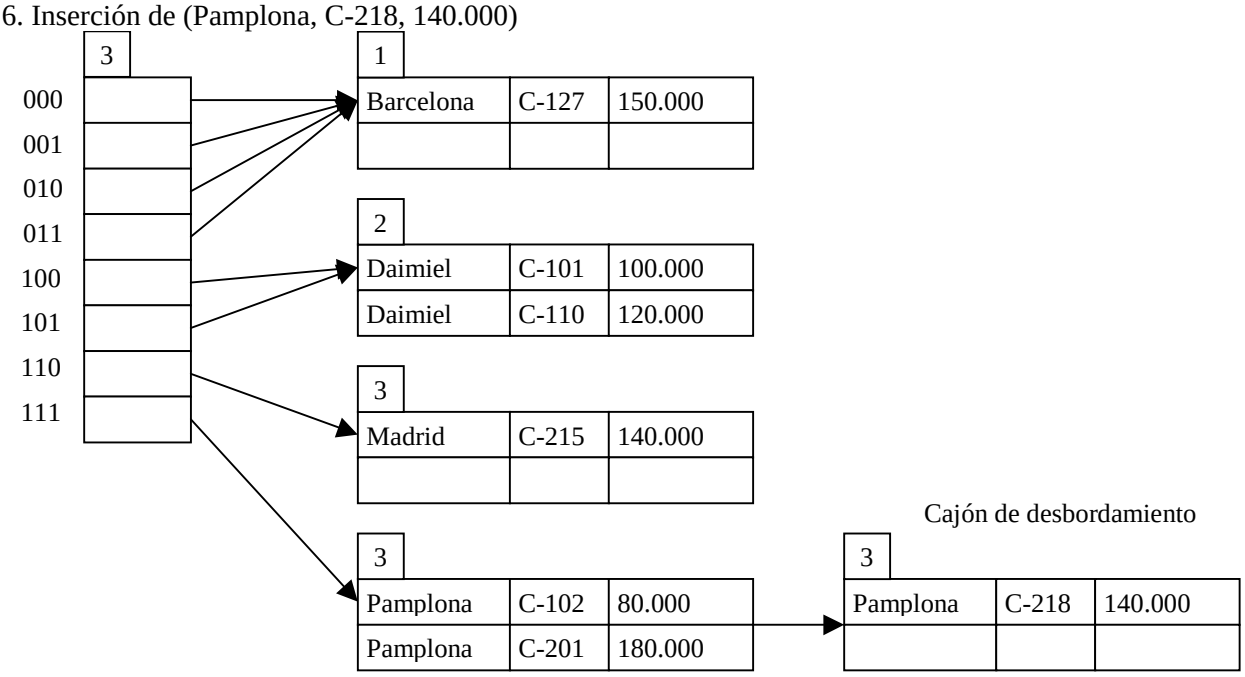
5. Inserción de (Pamplona, C-102, 80.000)



6. Inserción de (Pamplona, C-201, 180.000)



6. Inserción de (Pamplona, C-218, 140.000)



6.5. Operaciones sobre ficheros

6.5.1. Operaciones básicas

Indicadas en el apartado 6.1.2:

- Crear, borrar y modificar ficheros
- Acceso controlado a los ficheros de otros usuarios
- Reestructuración adecuada de los ficheros
- Movimiento de datos dentro del fichero
- Copias de seguridad y recuperación
- Acceso mediante nombres simbólicos

6.5.2. Filtrado

Función

Eliminación de campos de los registros.
Ej: En UNIX se usa cut
cut -d: -f1,5 /etc/passwd
-d: establece como delimitador de campos el carácter ":"
-f1,5 selecciona la lista (separada por comas) de los campos que deben aparecer en la salida
El resultado:
root:Super-User
adm:Admin
lp:Line Printer Admin
uucp:uucp Admin
nuucp:uucp Admin
listen:Network Admin
nobody:Nobody
noaccess:No Access User
nobody4:SunOS 4.x Nobody
manager:System Manager
juan:Juan Gómez
ramon:Ramón Sanjurjo
...

Implementación

Lectura secuencial del fichero por registro seguido de escritura secuencial con el registro procesado por el filtro.

6.5.3. Ordenación

Método de mezcla directa [Wir80]

En un dispositivo de acceso secuencial sólo se tiene acceso a un elemento de datos.
La técnica más importante es la de fusión o mezcla (merge), i.e., la combinación de dos o más secuencias ordenadas en una secuencia ordenada.
Mezcla directa:

1. Se divide la secuencia a en dos mitades: b y c.
2. Se mezclan b y c combinando elementos aislados para formar pares ordenados.
3. La secuencia resultante se denomina a y se repiten los pasos 1 y 2, mezclando los pares ordenados para formar cuádruplos ordenados.
4. Se repiten los pasos anteriores, mezclando los cuádruplos ordenados para formar óctuplos ordenados. Se continúa este procedimiento

Ejemplo:

44	55	12	42	94	18	6	67
b				c			

Paso 1:

b:	44	55	12	42
c:	94	18	6	67

Paso 2:

44	94	18	55	6	12	42	67
b				c			

Paso 1:

b:	44	94	18	55
c:	6	12	42	67

Paso 2:

6	12	44	94	18	42	55	67
b				c			

Paso 1:

b:	6	12	44	94
c:	18	42	55	67

Paso 2:

6	12	18	42	44	55	67	94

Bibliografía

[ACPT00] P. Atzeni, S. Ceri, S. Paraboschi y R. Torlone, “Database Systems. Concepts, Languages and Architectures”, McGraw-Hill, 2000.

[Dat93] C.J. Date, "Introducción a los Sistemas de Bases de Datos", Addison-Wesley, Reading Massachusetts, 1993.

[EN00] R. Elmasri y S.B. Navathe, "Fundamentals of Data Base Systems", Addison-Wesley, 2000.

[PLT97] A. Prieto, A. Lloris, J.C. Torres, "Introducción a la informática", 2ª edición, 1997. Apartado 6.1.6

[Sta97] W. Stallings, "Sistemas operativos", 2ª edición, Prentice Hall International, 1997. Apartados 6.1.1-6.1.3, 6.1.6, 6.4.1-6.4.4

[SKS98] A. Silberstachatz, …
Apartados 6.1.6-6.1.7,6.4

[Ull98] J.D. Ullman, "Principles of Database and Knowledge Base Systems", Vol. I y II, Computer Science Press, 1998.
Apartado Ficheros invertidos

[Wir80] N. Wirth, “Algoritmos + Estructuras de datos = Programas”, Ediciones del Castillo, 1980.
Apartado 6.5.3