

DISEÑO DE BASES DE DATOS RELACIONALES:

SISTEMAS GESTORES DE BASES DE DATOS

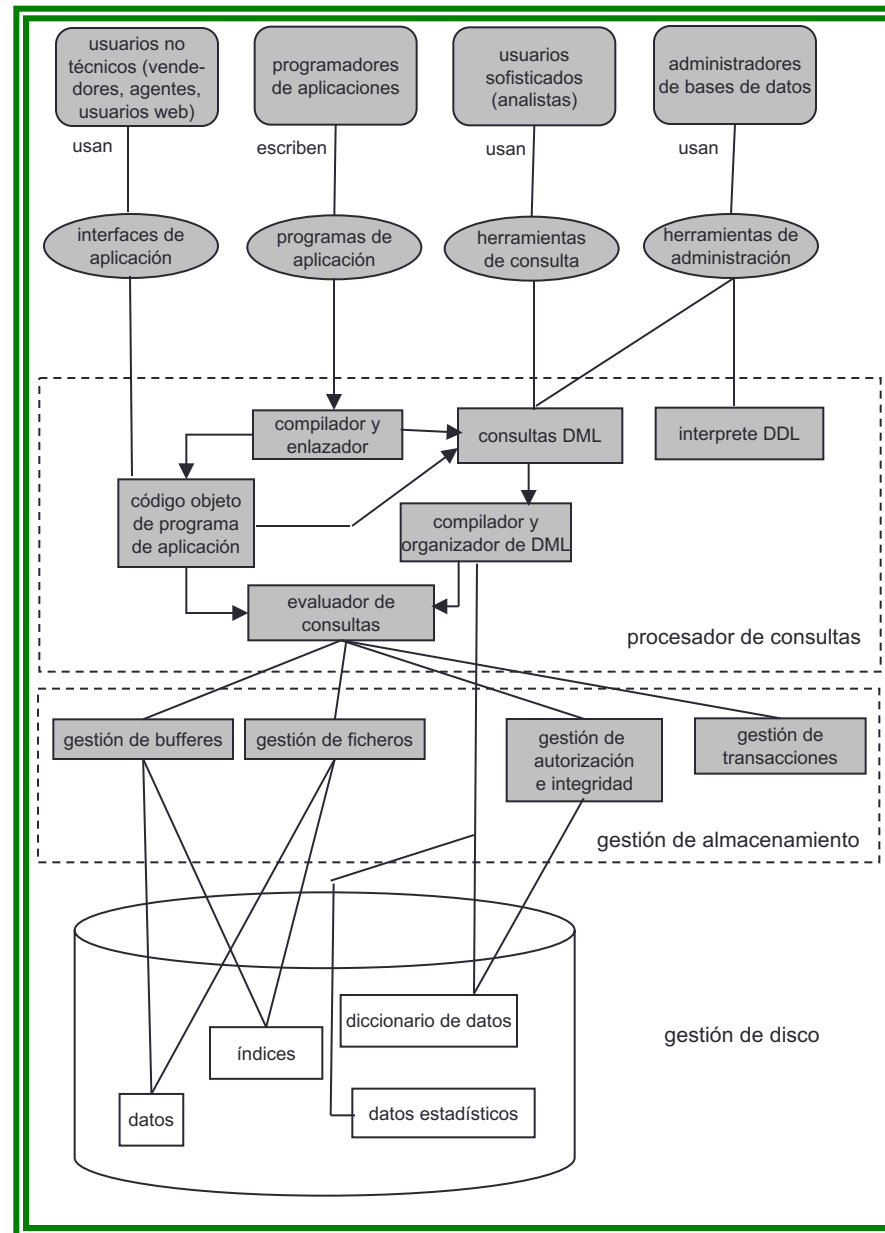
Tema 4

Manuel Ramos Cabrer

Índice

- Almacenamiento físico de los datos.
- Estructuras de ficheros
- Índices
- Asociaciones.
- Optimización de consultas.

SGBD



Clasificación de medios de almacenamiento físico

- Velocidad de acceso a los datos
- Coste por unidad de datos
- Fiabilidad
 - pérdida de datos ante fallos de alimentación y caídas del sistema
 - Fallos físicos del dispositivo de almacenamiento
- Podemos clasificar el almacenamiento en:
 - **almacenamiento volátil**: se pierde el contenido cuando se desconecta la alimentación
 - **almacenamiento no volátil**:
 - Los contenidos se mantienen aún cuando se desconecte la alimentación.
 - Incluye el almacenamiento secundario y terciario, así como la memoria principal con soporte de batería.

Medios de almacenamiento físico

- **Caché** – la forma de almacenamiento más rápida y costosa; volátil; gestionada por el hardware del sistema.
- **Memoria principal:**
 - acceso rápido (10s a 100s de nanosegundo; 1 nanosegundo = 10^{-9} segundos)
 - generalmente demasiado pequeña (o demasiado costosa) par almacenar la base de datos completa
 - normalmente se utilizan capacidades de unos pocos Gigabytes.
 - Las capacidades han crecido y los costes por byte han disminuido de manera constante y rápida (aproximadamente un factor de 2 cada 2 o 3 años)
 - **Volátil** — el contenido de la memoria principal normalmente se pierde si se produce un fallo de alimentación o una caída del sistema.

Medios de almacenamiento físico (Cont.)

- **Memoria flash**

- Los datos se mantienen ante un fallo de alimentación
- Los datos se pueden escribir una sola vez en una posición, pero una posición se puede borrar y escribir de nuevo
 - Pueden soportar sólo un número limitado (10K – 1M) de ciclos de escritura/borrado.
 - El borrado se tiene que hacer sobre bancos enteros de memoria
- La lectura es aproximadamente tan rápida como la de memoria principal
- Pero la escritura es lenta (pocos microsegundos), el borrado es aún más lento
- El coste por unidad es similar al de la memoria principal
- Ampliamente utilizado en dispositivos embebidos tales como cámaras digitales, teléfonos móviles, y llaves USB.

Medios de almacenamiento físico (Cont.)

- **Discos magnéticos**

- Los datos se almacenan en discos giratorios, y que se leen/escriben magnéticamente
- Es el principal medio de almacenamiento de datos a largo plazo; típicamente almacenan bases de datos enteras.
- Los datos se deben mover de disco a memoria principal para acceder a ellos, y escritos de nuevo hacia el disco si se modifican
 - Los accesos son mucho más lentos que a memoria principal
- **Acceso directo** – es posible leer datos de disco en cualquier orden, al contrario que en las cintas magnéticas
- **Discos duros** vs **discos flexibles**
- Las capacidades actuales llegan hasta más de 10 TB
 - Mucha mayor capacidad y coste/byte que la memoria principal/flash
 - Crece de manera constante y rápida con las mejoras tecnológicas (factor de 2 a 3 cada 2 años)
- Sobrevive a fallos de corriente y caídas del sistema
 - los fallos de disco pueden destruir datos, pero son muy infrecuentes

Medios de almacenamiento físico (Cont.)

- **Almacenamiento óptico**

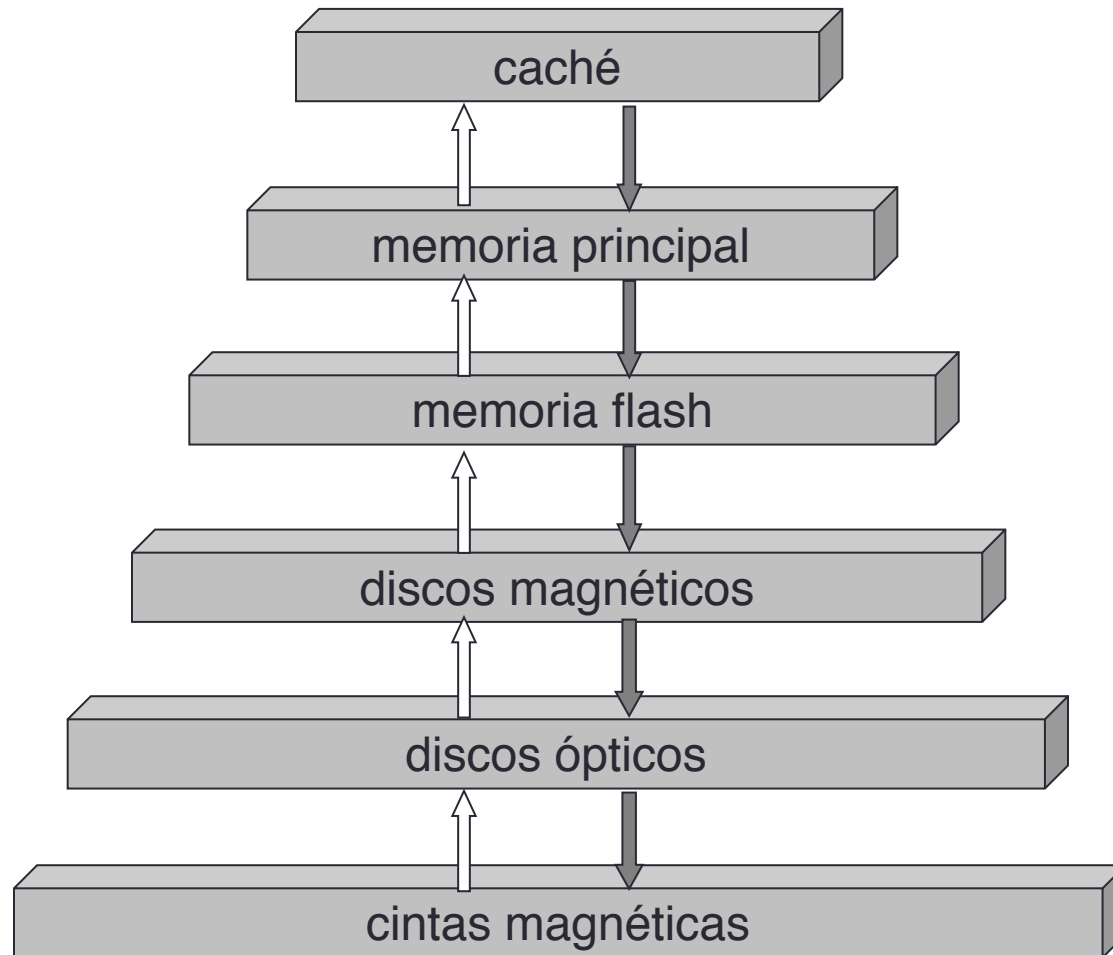
- no-volátil, los datos se leen ópticamente de un disco giratorio utilizando un láser
- Los medios más comunes son CD-ROM (640 MB) y DVD (4.7 a 17 GB)
- Discos Blu-ray: 27 a 54 GB.
- Los discos ópticos de una-escritura, muchas lecturas (WORM) se utilizan para almacenamiento de archivos (CD-R y DVD-R)
- También existen versiones de múltiples escrituras (CD-RW, DVD-RW, y DVD-RAM)
- Las lecturas y escrituras son más lentas que en los discos magnéticos
- **Juke-box**: son sistemas con un número elevado de discos extraíbles, unos pocos lectores y un mecanismo para la carga y descarga automática de discos. Permiten el almacenamiento de grandes volúmenes de datos

Medios de almacenamiento físico (Cont.)

- **Almacenamiento en cinta**

- no volátil, se utiliza principalmente para copias de seguridad (recuperación frente a fallos de disco) y para archivo de datos
- **Acceso secuencial** – mucho más lento que el disco
- capacidad muy alta (hay cintas de 40 a 300 GB)
- las cintas se pueden extraer del dispositivo
- el almacenamiento es mucho más barato que en disco, pero los dispositivos son caros
- Existen jukeboxes de cintas para el almacenamiento de cantidades masivas de datos
 - cientos de terabytes (1 terabyte = 10^{12} bytes) hasta varios petabyte (1 petabyte = 10^{15} bytes)

Jerarquía de almacenamiento



Jerarquía de almacenamiento (Cont.)

- **Almacenamiento primario**: Medio más rápido pero volátil (caché, memoria principal).
- **Almacenamiento secundario**: siguiente nivel en la jerarquía, no volátil, tiempo de acceso moderado
 - también llamado **almacenamiento en línea**
 - P.e. Memoria flash, discos magnéticos
- **Almacenamiento terciario**: menor nivel en la jerarquía, no volátil, tiempo de acceso lento
 - también llamado **almacenamiento fuera de línea**
 - P.e. Cintas magnéticas, almacenamiento óptico

Medidas de fiabilidad

- **Tiempo medio entre fallos (MTTF)** – el tiempo medio que se espera que el disco funcione de manera continua sin ningún fallo.
 - Valores normales: 3 a 5 años
 - La probabilidad de fallo de discos nuevos es bastante baja, correspondiendo a un “MTTF teórico” de 30.000 a 1.200.000 horas para un disco nuevo
 - P.e., un MTTF de 1,200,000 horas para un disco nuevo significa que dados 1.000 discos relativamente nuevos, en media uno fallara cada 1.200 horas
 - El MTTF se decrementa con la edad de los discos.

RAID en un SGBD

- **RAID: Redundant Arrays of Independent Disks**

- Técnicas de organización de disco que gestionan una gran cantidad de discos, proporcionando la imagen de un solo disco de
 - **gran capacidad** y **alta velocidad** utilizando varios discos en paralelo, y
 - **alta disponibilidad** almacenando datos de forma redundante, de tal manera que los datos se pueden recuperar aún cuando un disco falle
- La probabilidad de que algún disco de un conjunto de N discos falle es mucho mayor que la probabilidad de que un único disco falle.
 - P.e., un sistema con 100 discos, cada uno con un MTTF de 100.000 horas (aprox. 11 años), tendrá un MTTF de 1.000 horas (aprox. 41 días)
 - El uso de técnicas que utilicen redundancia para evitar pérdidas de datos es crítico si utilizamos un número elevado de discos
- Originalmente era una alternativa de bajo coste a tener discos grandes y caros
 - La I de RAID originalmente significaba “inexpensive”
 - Hoy en día los RAIDs se utilizan por su alta disponibilidad y ancho de banda.
 - La “I” significa “independent”

Aumento de la disponibilidad mediante redundancia

- **Redundancia** – guardar información extra que se puede utilizar para reconstruir información perdida en un fallo de disco
- P.e., **Discos espejo** (o *shadowing*)
 - Duplicar cada disco. Los discos lógicos están formados por dos discos físicos.
 - Cada escritura se realiza en ambos discos
 - Las lecturas se pueden realizar de cualquier disco
 - Si uno de los dos discos falla, los datos están disponibles en el otro
 - Sólo se perderán datos si un disco falla, y su disco espejo también falla antes de que se hubiera reparado el primero
 - La probabilidad de este evento combinado es muy baja
 - Excepto para fallos dependientes entre sí tales como incendio, fallo de alimentación ,...
- El **tiempo medio entre pérdida de datos** depende del tiempo medio entre fallos y el **tiempo medio de reparación**
 - P.e. Un MTTF de 100.000 horas y un tiempo medio de reparación de 10 horas da un tiempo medio entre pérdidas de datos de $500 \cdot 10^6$ horas (o 57.000 años) para discos espejo (ignorando los fallos dependientes)

“Hot swap”

- **Cambio en caliente:** sustitución de discos mientras el sistema está en ejecución, sin apagado.
 - Soportado por algunos sistemas RAID hardware,
 - Reduce el tiempo de recuperación y aumenta mucho la disponibilidad
- Muchos sistemas mantienen **discos de repuesto** que se mantienen en línea y se utilizan para reemplazar discos que fallan de manera inmediata tras la detección del fallo
 - Reduce mucho el tiempo de recuperación
- Muchos sistemas RAID hardware consiguen que un solo punto de fallo no detenga el funcionamiento del sistema utilizando
 - Fuentes de alimentación redundantes con respaldo de baterías.
 - Varios controladores y varias conexiones para protegerse frente a fallos de controlador/interconexión.

Discos Ópticos

- CD-ROM (Compact disk-read only memory)
 - Discos que se pueden cargar y descargar de un dispositivo
 - Gran capacidad (640 MB por disco)
 - Tiempos de posicionamiento altos de alrededor de 100 msg. (la cabeza de lectura óptica es más pesada y lenta que la de un disco duro)
 - Latencia más alta (3000 RPM) y tasas de transferencia peores (3-6 MB/s) que las de un disco magnético
- DVD (Digital Video Disk)
 - DVD-5 almacena 4.7 GB , y DVD-9 almacena 8.5 GB
 - DVD-10 y DVD-18 son formatos de doble cara con capacidades de 9.4 GB y 17 GB
 - DVD Blu-ray: 27 GB (54 GB en discos de doble cara)
 - Los tiempos de posicionamiento son similares a las de un CD-ROM
- CD-R y DVD-R hoy en día bastante utilizados (una sola escritura)
 - Los datos se pueden escribir una sola vez y no se pueden borrar.
 - Gran capacidad y larga vida; se utilizan como archivo
 - También existen versiones de varias escrituras (CD-RW, DVD-RW y DVD-RAM)

Cintas Magnéticas

- Almacenan grandes cantidades de información y proporcionan altas tasas de transferencia
 - Unos pocos GB en el formato DAT (Digital Audio Tape), 10-40 GB en el formato DLT (Digital Linear Tape), 100 GB+ en el formato Ultrium, y 330 GB en el formato helicoidal Ampex.
 - Las tasas de transferencia son de unas pocas 10s de MB/sg.
- Actualmente es el medio de almacenamiento más barato
 - Las cintas son baratas, pero los dispositivos son muy caros.
- Tiempos de acceso muy lentos en comparación con los discos magnéticos y los discos ópticos
 - Limitadas a acceso secuencial.
 - Algunos formatos (Accelis) proporciona búsquedas más rápidas (10s de segundo) a cambio de menor capacidad.
- Se usan principalmente para copias de seguridad, para almacenar información que no se utiliza frecuentemente, y como un medio no interactivo de transferir información de un sistema a otro.
- Los robots de cintas (*jukeboxes*) se utilizan para conseguir capacidades de almacenamiento muy elevadas
 - del terabyte (10^{12} bytes) al petabyte (10^{15} bytes)

Organización de ficheros

- La base de datos se almacena como un conjunto de ***ficheros***.
- Cada fichero es una secuencia de ***registros***.
- Un registro es una secuencia de ***campos***.
- Una aproximación:
 - suponer que el tamaño de registro es fijo.
 - cada fichero almacena solamente registros de un tipo concreto.
 - se utilizan ficheros distintos para relaciones distintas.

Este caso es el más sencillo de implementar.

Registros de tamaño fijo

- Aproximación simple:
 - Guardar el registro i empezando en el byte $n * (i - 1)$, donde n es el tamaño de cada registro.
 - El acceso a registros es sencillo pero los registros pueden cruzar bloques
 - Modificación: no permitir que los registros crucen bloques
- Borrado del registro i : alternativas:
 - Mover registros $i + 1, \dots, n$ a $i, \dots, n - 1$
 - Mover registro n a i
 - No mover registros, pero enlazar todos los registros libres en una *lista de huecos*

registro 0	10101	Samuel	Ing. Telemática	65000
registro 1	12121	Vicente	Economía	90000
registro 2	15151	Manuel	Música	40000
registro 3	22222	Emilio	Física	95000
registro 4	32343	Elena	Historia	60000
registro 5	33456	Gabriela	Física	87000
registro 6	45565	Carla	Ing. Telemática	75000
registro 7	58583	Carlos	Historia	62000
registro 8	76543	Sandra	Economía	80000
registro 9	76766	Cristina	Biología	72000
registro 10	83821	Blanca	Ing. Telemática	92000
registro 11	98345	Ana	Tec. Electrónica	80000

Eliminar el registro 3 y compactar

registro 0	10101	Samuel	Ing. Telemática	65000
registro 1	12121	Vicente	Economía	90000
registro 2	15151	Manuel	Música	40000
registro 4	32343	Elena	Historia	60000
registro 5	33456	Gabriela	Física	87000
registro 6	45565	Carla	Ing. Telemática	75000
registro 7	58583	Carlos	Historia	62000
registro 8	76543	Sandra	Economía	80000
registro 9	76766	Cristina	Biología	72000
registro 10	83821	Blanca	Ing. Telemática	92000
registro 11	98345	Ana	Tec. Electrónica	80000

Eliminar el registro 3 y mover el último

registro 0	10101	Samuel	Ing. Telemática	65000
registro 1	12121	Vicente	Economía	90000
registro 2	15151	Manuel	Música	40000
registro 11	98345	Ana	Tec. Electrónica	80000
registro 4	32343	Elena	Historia	60000
registro 5	33456	Gabriela	Física	87000
registro 6	45565	Carla	Ing. Telemática	75000
registro 7	58583	Carlos	Historia	62000
registro 8	76543	Sandra	Economía	80000
registro 9	76766	Cristina	Biología	72000
registro 10	83821	Blanca	Ing. Telemática	92000

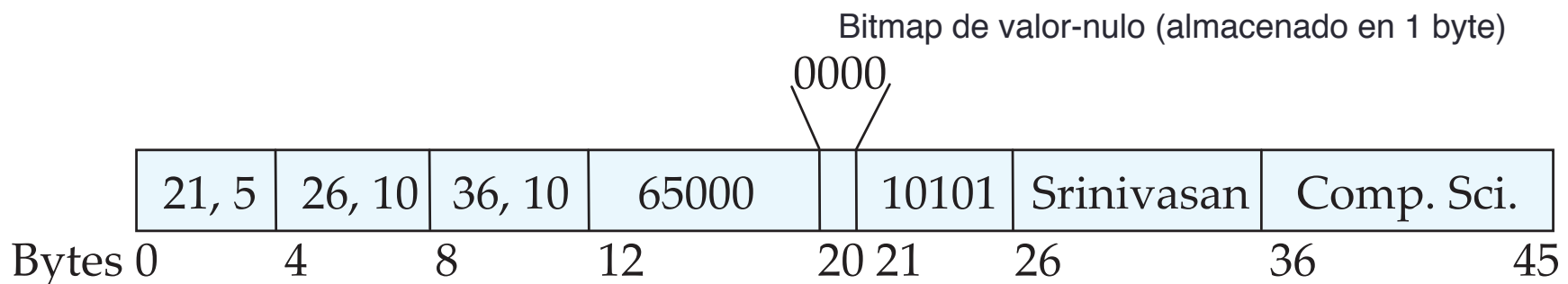
Lista de huecos

- Almacena la dirección del primer registro borrado en la cabecera de fichero.
- Utiliza el primer registro para almacenar la dirección del segundo registro borrado, y así sucesivamente
- Podemos pensar en esas direcciones almacenadas como **punteros** dado que “apuntan” a la ubicación del registro.
- Representación más eficiente en cuanto a espacio: reutilizar el espacio de los atributos normales de los registros libres para almacenar los punteros (No hay que guardar punteros en los registros que se están utilizando)

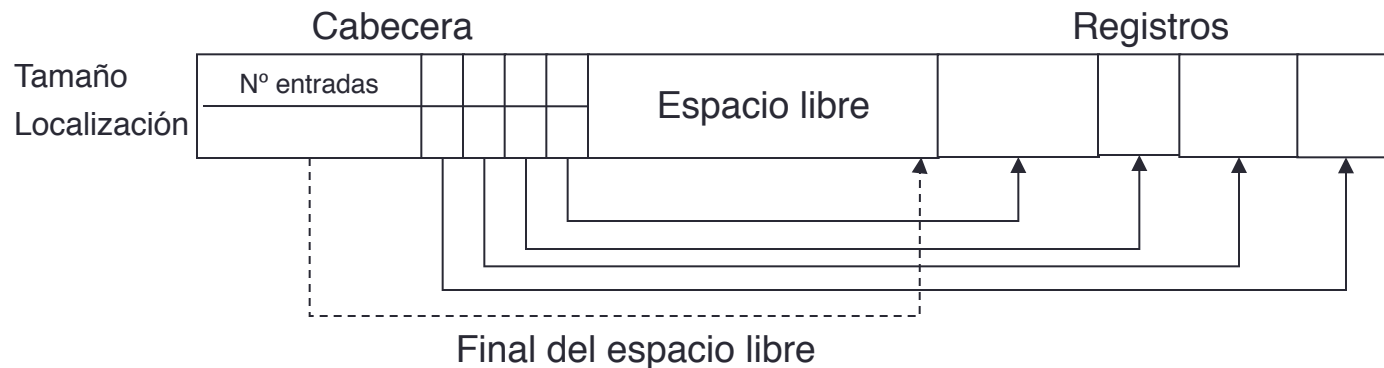
cabecera				
registro 0	1010 1	Samuel	Ing. Telemática	65000
registro 1				
registro 2	1515 1	Manuel	Música	40000
registro 3	2222 2	Emilio	Física	95000
registro 4				
registro 5	3345 6	Gabriela	Física	87000
registro 6				
registro 7	5858	Carlos	Historia	62000

Registros de tamaño variable

- Los registros de tamaño variable aparecen en las bases de datos por diversos motivos:
 - Almacenamiento de varios tipos de registros en el mismo fichero.
 - Tipos de registros que permiten tamaños variables en uno o más campos, tales como strings (**varchar**).
 - Tipos de registros que permiten campos repetitivos (se utilizan en algunos modelos de datos antiguos).
- Los atributos se almacenan en orden
- Los atributos de longitud variable se representan con entradas de tamaño fijo (offset, longitud), con el dato real almacenado después de todos los atributos de tamaño fijo.
- Los valores null se representan mediante un bitmap de “valor-nulo”



Registros de tamaño variable: Estructura de página ranurada



- La cabecera de una **página ranurada** contiene:
 - el número de registros
 - El final del espacio libre de la página
 - La localización y tamaño de cada registro
- Los registros se pueden mover dentro de la página para mantenerlos contiguos sin espacio libre entre ellos; se debe actualizar la entrada de la cabecera.
- Los punteros no deberían apuntar directamente a los registros — deberían apuntar a la entrada en la cabecera para el registro.

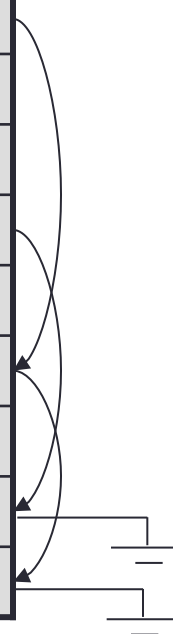
Registros de tamaño variable (Cont.)

- Representación de tamaño fijo:
 - espacio reservado
 - punteros
- **Espacio reservado** – se pueden utilizar registros de tamaño fijo de un tamaño máximo conocido; el espacio no utilizado en registros más pequeños se rellena con un valor nulo o símbolo de fin-de-registro.

0	Ing. Telemática	EE de Teleco	40000	12121	22222	54321	55555
1	Historia	Fac. Historia	35000	34567	⊥	⊥	⊥
2	Física	Fac. Ciencias	70000	20002	⊥	⊥	⊥
3	Biología	Fac. Ciencias	50000	78912	98765	⊥	⊥
4	Tec. Electrónica	EE de Teleco	70000	11111	12312	56565	⊥
5	Economía	Fac. Economía	75000	⊥	⊥	⊥	⊥

Método con punteros

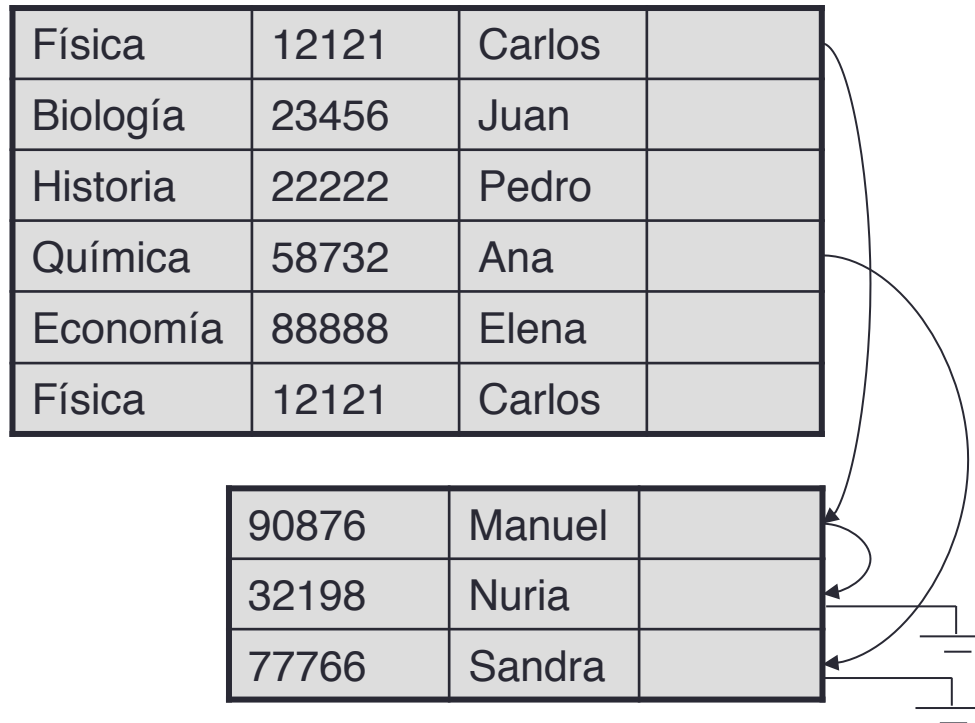
0	Física	12121	Carlos	
1	Biología	23456	Juan	
2	Historia	22222	Pedro	
3	Química	58732	Ana	
4	Economía	88888	Elena	
5		90876	Manuel	
6	Música	33344	Gabriel	
7		77766	Sandra	
8		32198	Nuria	



- **Método con punteros**
 - Un registro de longitud variable se representa mediante una lista de registros de tamaño fijo, encadenados juntos mediante punteros.
 - Se puede utilizar aun cuando no se conozca el tamaño máximo de registro.

Método con punteros (Cont.)

- Desventaja de la estructura con punteros: se malgasta espacio en todos los registros salvo el primero de una cadena.
- Solución: permitir dos clases de bloques en un fichero:
 - **Bloques principales** – contienen el primer registro de cada cadena
 - **Bloques de desbordamiento** – contienen los demás registros de cada cadena




Organización de registros en ficheros

- **Pila** – un registro se puede colocar en cualquier lugar del fichero en que haya sitio.
- **Secuencial** – los registros se almacenan de manera secuencial, en base al valor de la clave de búsqueda de cada registro.
- **Hashing** – se calcula una función hash sobre algún atributo de cada registro; el resultado indica en que bloque de cada fichero se colocará el registro.
- Los registros de cada relación se pueden guardar en un fichero separado. En una **organización de ficheros en cluster** los registros de varias relaciones diferentes se pueden almacenar en el mismo fichero.
 - Motivación: almacenar los registros relacionados en el mismo bloque para minimizar la E/S.

Organización de fichero secuencial

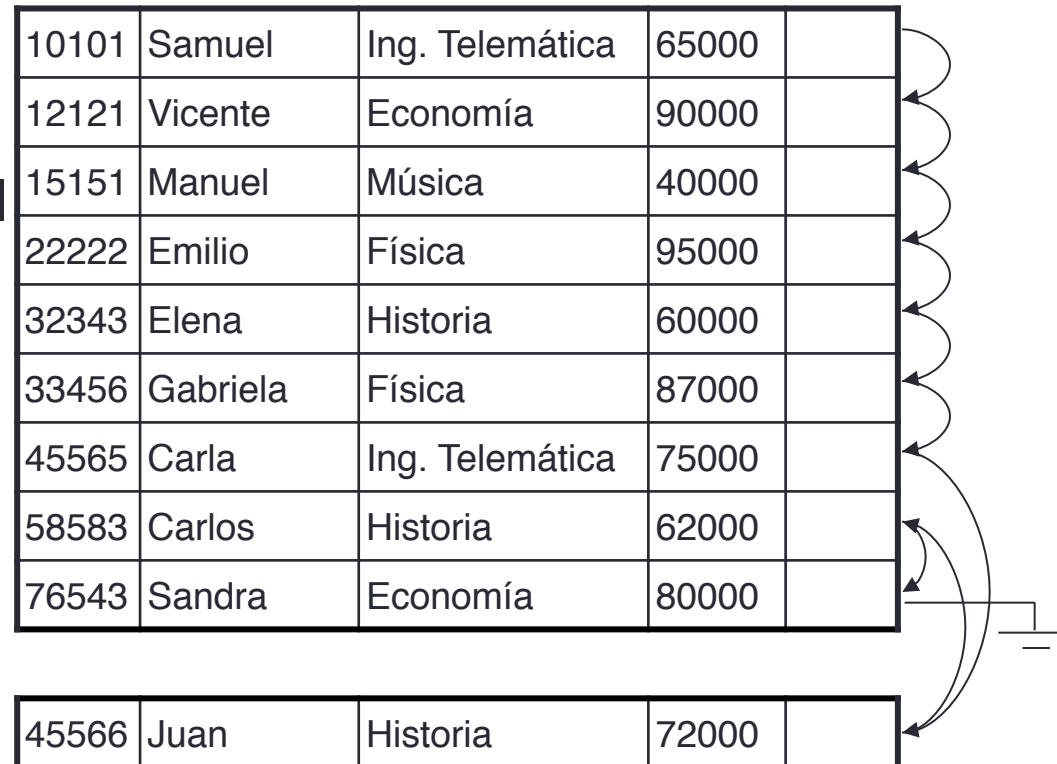
- Adecuada para aplicaciones que requieren un procesamiento secuencial del fichero completo
- Los registros están ordenados en el fichero por una **clave de búsqueda**

10101	Samuel	Ing. Telemática	65000	
12121	Vicente	Economía	90000	
15151	Manuel	Música	40000	
22222	Emilio	Física	95000	
32343	Elena	Historia	60000	
33456	Gabriela	Física	87000	
45565	Carla	Ing. Telemática	75000	
58583	Carlos	Historia	62000	
76543	Sandra	Economía	80000	



Organización de fichero secuencial(Cont.)

- Borrado – se utiliza la cadena de punteros
- Inserción – se localiza la posición donde se inserta el registro
 - Si hay espacio libre se inserta allí
 - Si no hay espacio libre, se inserta el registro en un **bloque de desbordamiento**
 - En cualquier caso, se debe actualizar la cadena de punteros
- Se necesita reorganizar el fichero de vez en cuando para recuperar el orden secuencial



Organización de fichero en cluster

- La estructura de fichero simple almacena cada relación en un fichero distinto
- En vez de ello, se pueden almacenar varias relaciones en un fichero utilizando una organización de ficheros en **cluster**
- P.e., organización en cluster de *docente* y *departamento*:

Departamento

<i>Nombre_dpto</i>	<i>Edificio</i>	<i>Presupuesto</i>
Ing. Telemática	EE de Teleco	100000
Física	Fac. Ciencias	70000

Docente

<i>ID</i>	<i>Nombre</i>	<i>Nombre_dpto</i>	<i>Salario</i>
45564	Carlos	Ing. Telemática	75000
10101	Silvia	Ing. Telemática	65000
83821	Benito	Ing. Telemática	92000
33456	Gabriela	Física	87000

Cluster multitable de docente y departamento

Ing. Telemática	EE de Teleco	100000
45564	Carlos	75000
10101	Silvia	65000
83821	Benito	92000
Física	Fac. Ciencias	70000
33456	Gabriela	87000

Organización de fichero en cluster

Ing. Telemática	EE de Teleco	100000	
45564	Carlos	75000	
10101	Silvia	65000	
83821	Benito	92000	
Física	Fac. Ciencias	70000	
33456	Gabriela	87000	



- buena para consultas que afecten a *docente* ⋈ *departamento*, y para consultas que afecten a un solo departamento y sus docentes
- mala para consultas que afecten sólo a departamentos
- da lugar a registros de tamaño variable
- pueden añadirse cadenas de punteros para enlazar registros de una determinada relación

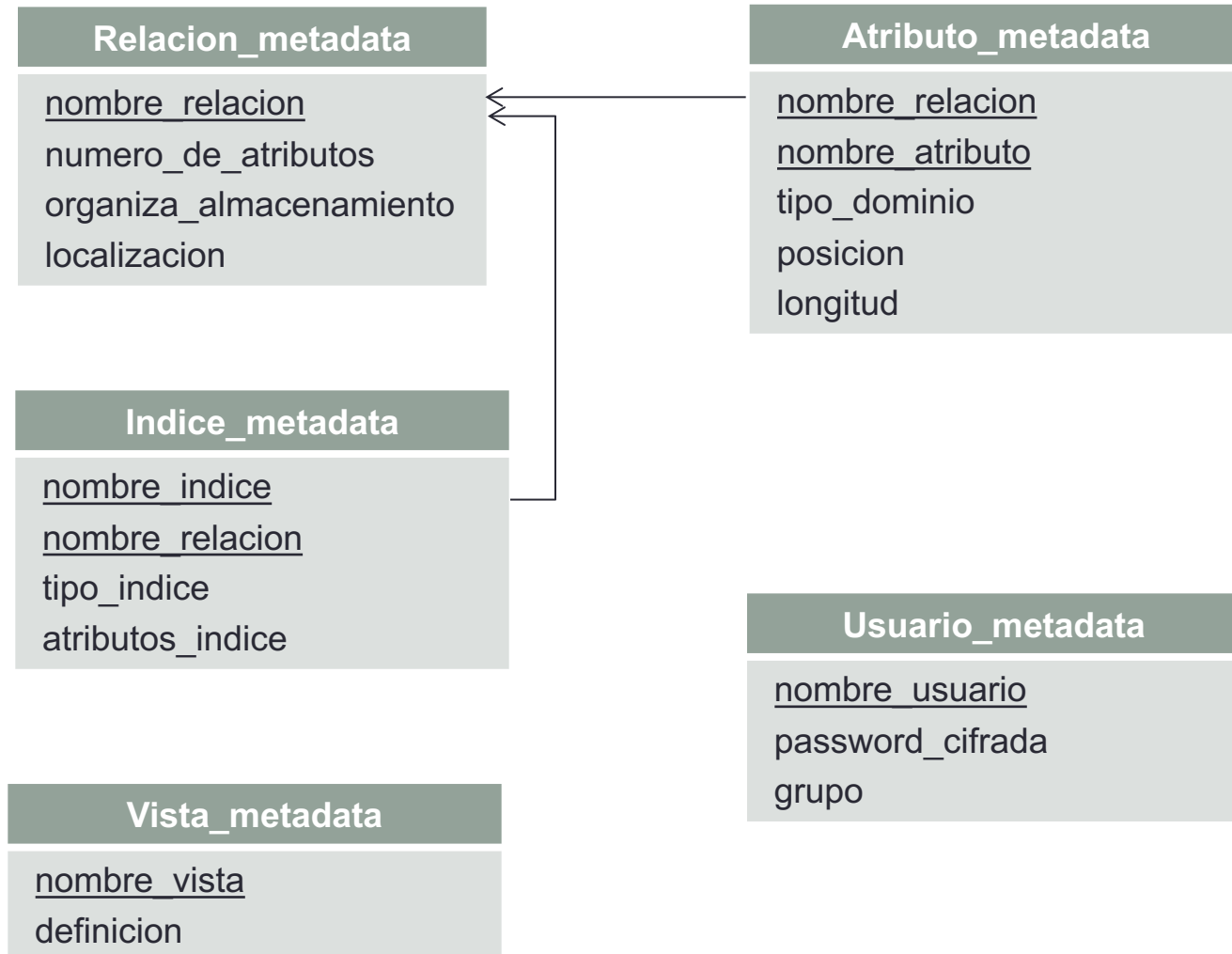
Almacenamiento del diccionario de datos

El **diccionario de datos** (también llamado **catálogo del sistema**) almacena **metadatos**, es decir, datos sobre los datos, tales como:

- Información sobre las relaciones
 - nombres de las relaciones
 - Nombres y tipos de los atributos de cada relación
 - nombres y definiciones de vistas
 - restricciones de integridad
- Información de usuarios y cuentas, incluyendo passwords
- Datos estadísticos y descriptivos
 - número de tuplas en cada relación
- Información sobre la organización física de ficheros
 - cómo se almacena una relación (secuencial/hash/...)
 - localización física de la relación
 - nombre del fichero en el sistema operativo o
 - direcciones en disco de los bloques que contienen los registros de la relación
- Información sobre índices

Almacenamiento del diccionario de datos (Cont.)

- Estructura del catálogo: puede ser
 - o bien estructuras de datos especializadas diseñadas para un acceso eficiente
 - o bien un conjunto de relaciones, utilizando las características propias del sistema para asegurar un uso eficiente
- Se suele utilizar la segunda alternativa



Acceso al almacenamiento

- Un fichero de base de datos se divide en unidades de almacenamiento de tamaño fijo denominadas **bloques**. Los bloques son unidades tanto de asignación de almacenamiento como de transferencia de datos.
- Un sistema de bases de datos debe tratar de minimizar el número de transferencias de bloques entre disco y memoria. Se puede reducir el número de accesos a disco manteniendo en la memoria principal tantos bloques como sea posible.
- **Bufer** – porción de la memoria principal disponible para almacenar copias de bloques de disco.
- **Gestor de bufer** – subsistema responsable de asignar espacio de bufer en memoria principal.

Gestión de bufer

- Los programas llaman al gestor de bufer cuando necesitan un bloque del disco.
 1. Si el bloque ya está en el bufer, se devuelve al programa la dirección del bloque en memoria principal.
 2. Si el bloque no está en el bufer,
 1. El gestor de bufer asigna espacio en el bufer para el bloque, reemplazando (desalojando) otro bloque si es necesario para hacer espacio para el nuevo bloque.
 2. El bloque que se desaloja se escribe de nuevo al disco solo si se ha modificado desde la última vez que se escribió/leyó del disco.
 3. Una vez se dispone de espacio en el bufer, el gestor lee el bloque de disco al bufer, y devuelve al programa la dirección del bloque en memoria principal.

Políticas de sustitución

- La mayoría de los sistemas operativos reemplazan los bloques **utilizados menos recientemente** (estrategia LRU)
- La idea de LRU es – utilizar el patrón pasado de referencias a bloque como predicción de referencias futuras.
- Las consultas tienen patrones de acceso bien definidos (tal como lecturas secuenciales), y un sistema de bases de datos puede utilizar la información en una consulta de usuario para predecir referencias futuras
 - LRU puede ser una estrategia mala para ciertos patrones de acceso que supongan accesos repetitivos a datos.
 - P.e. cuando se calcula el join de 2 relaciones r y s mediante bucles anidados para cada tupla tr de r hacer
 - para cada tupla ts de s hacer
 - si las tuplas tr y ts coinciden ...
 - Es preferible utilizar estrategias mixtas proporcionadas por el optimizador de consultas

Políticas de sustitución (Cont.)

- **Bloques anclados** – bloques de memoria que no se pueden trasegar al disco.
- Estrategia del **lanzamiento inmediato** – libera el espacio ocupado por un bloque tan pronto como la última tupla de ese bloque se ha procesado.
- Estrategia del **utilizado más recientemente (MRU)** – el sistema debe marcar el bloque que se está procesando. Después de procesar la última tupla del bloque se desmarca y pasa a ser el bloque utilizado más recientemente.
- La gestión de bufer puede utilizar información estadística considerando la probabilidad de que una petición referencie en el futuro a una determinada relación
 - P.e., el diccionario de datos se accede frecuentemente.
Heurístico: mantener los bloques del diccionario de datos en memoria principal.

INDEXADO Y ASOCIACIÓN

Conceptos básicos

- Los índices se utilizan para acelerar el acceso a los datos deseados.
 - P.e., catálogo de autores en una biblioteca
- **Clave de búsqueda** – atributo utilizado para buscar registros en un fichero.
- Un **fichero índice** está formado por registros (denominados **entradas del índice**) de la forma

clave-busqueda	puntero
----------------	---------

- Los ficheros índice suelen ser mucho más pequeños que los ficheros originales
- Los tipos básicos de índices:
 - **Índices ordenados:** las claves de búsqueda se almacenan ordenadas
 - **Índices hash:** las claves de búsqueda se distribuyen de manera uniforme entre “cajones” utilizando una “función hash”.

Métricas de evaluación de índices

- Tipos de accesos soportados de manera eficiente. P.e.,
 - registros con un valor especificado en el atributo
 - o registros con un valor del atributo dentro de un determinado rango de valores.
- Tiempo de acceso
- Tiempo de insercción
- Tiempo de borrado
- Sobrecarga de espacio

Índices ordenados

Técnicas de indexado evaluadas en base a:

- En un **índice ordenado**, las entradas del índice se almacenan ordenadas por el valor de la clave de búsqueda. P.e., catálogo de autores en una biblioteca.
- **Índice primario**: en un fichero ordenado secuencialmente, el índice cuya clave de búsqueda especifica el orden secuencial del fichero.
 - También denominado **índice cluster**
 - La clave de búsqueda de un índice primario es usualmente, pero no necesariamente, la clave primaria.
- **Índice secundario**: un índice cuya clave de búsqueda especifica un orden diferente del orden secuencial del fichero. También denominado **índice no cluster**.
- **Fichero índice secuencial**: fichero ordenado secuencialmente con un índice primario.

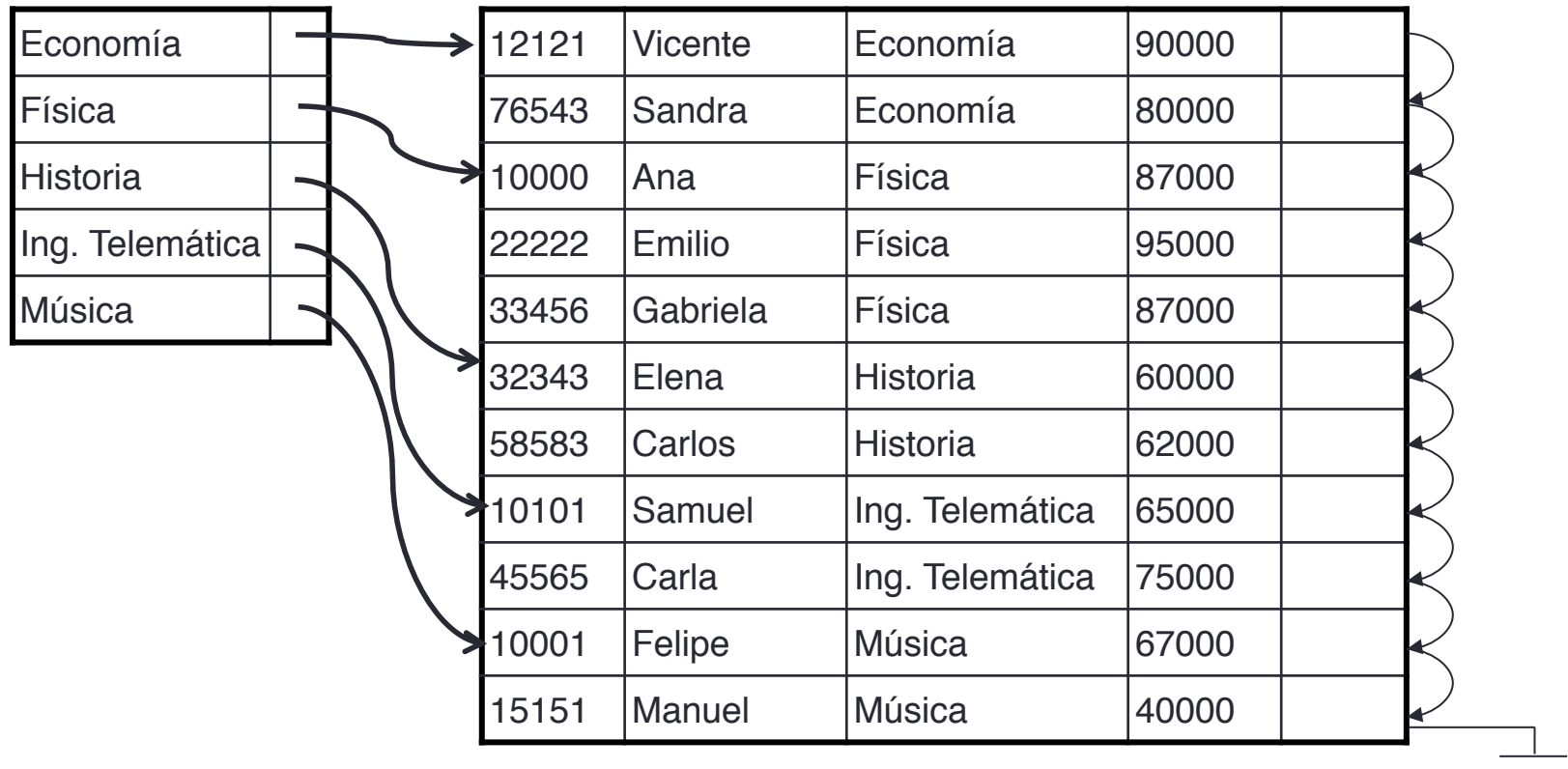
Ficheros índice densos

- **Índice denso** — Hay un registro índice por cada valor de la clave de búsqueda en el fichero.
- P.e. índice sobre el atributo *ID* en la relación *docente*.

10000		→	10000	Ana	Física	87000	
10001		→	10001	Felipe	Música	67000	
10101		→	10101	Samuel	Ing. Telemática	65000	
12121		→	12121	Vicente	Economía	90000	
15151		→	15151	Manuel	Música	40000	
22222		→	22222	Emilio	Física	95000	
32343		→	32343	Elena	Historia	60000	
33456		→	33456	Gabriela	Física	87000	
45565		→	45565	Carla	Ing. Telemática	75000	
58583		→	58583	Carlos	Historia	62000	
76543		→	76543	Sandra	Economía	80000	

Ficheros índice densos

- Índice denso sobre *nombre_dpto*, con el fichero *docente* ordenado por *nombre_dpto*.



Ficheros índice dispersos

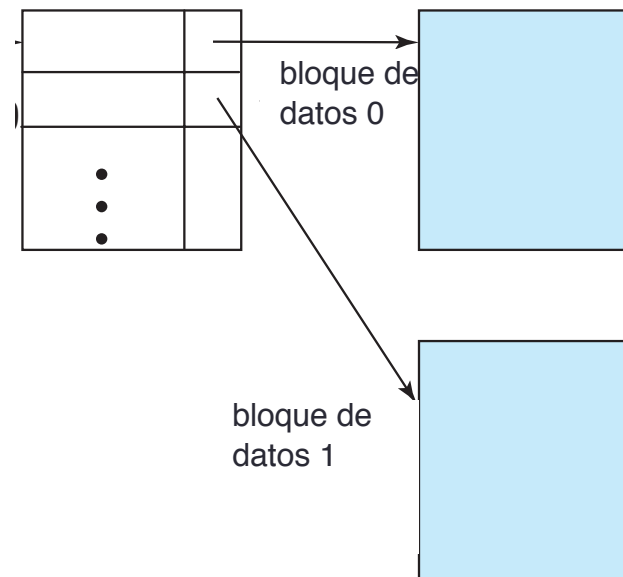
- **Ficheros dispersos**: contienen registros índice sólo para algunos valores de la clave de búsqueda.
 - Aplicable cuando los registros están ordenados secuencialmente sobre la clave de búsqueda
- Para localizar un registro con valor de la clave de búsqueda K se debe:
 - Encontrar el registro índice con el mayor valor de la clave de búsqueda $< K$.
 - Buscar en el fichero secuencialmente empezando en el registro al que apunta el registro índice.

Ejemplo de fichero índice disperso



Ficheros índice dispersos (Cont.)

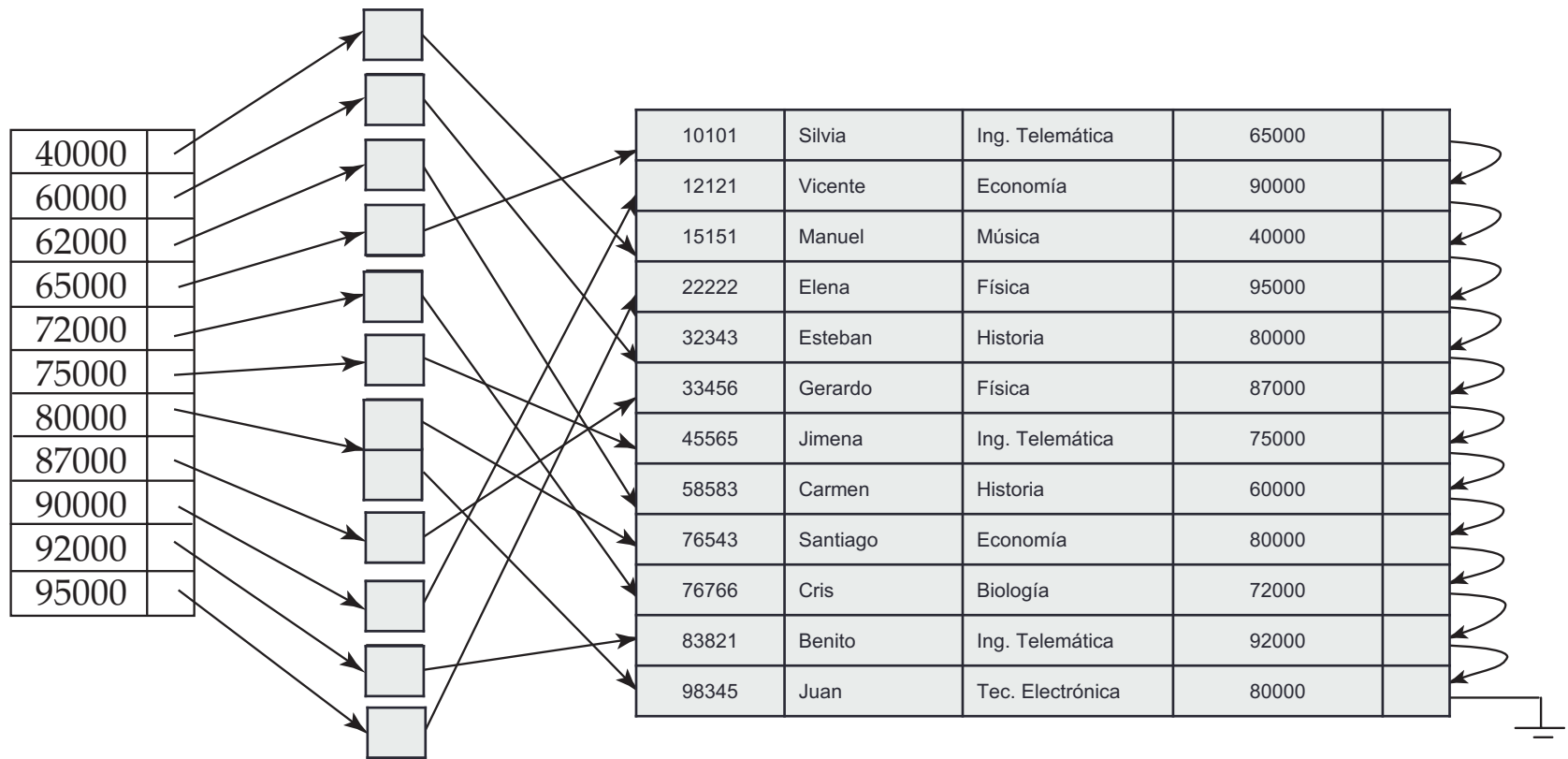
- Comparado con índices densos:
 - Menos espacio y menos sobrecarga de mantenimiento para inserciones y eliminaciones.
 - Generalmente más lentos que los índices densos para encontrar registros.
- **Buen balance:** un índice dispersos con una entrada de índice para cada bloque en el fichero, correspondiente al valor inferior de la clave de búsqueda almacenada ese bloque.



Índices secundarios

- A menudo se quieren encontrar todos los registros cuyos valores en un cierto campo (que no es la clave de búsqueda del índice primario) cumplen una determinada condición.
 - Ejemplo 1: En la relación *docentes* almacenada secuencialmente por el identificador de docente, podemos querer encontrar todos los docentes de un determinado departamento.
 - Ejemplo 2: como antes, pero queremos encontrar todas los docentes con un salario determinado o dentro de un rango de salarios.
- Podemos tener un índice secundarios con un registro índice para cada valor de la clave de búsqueda.

Ejemplo de índice secundario



- Los registros índice apuntan a unos “cajones” que contienen punteros a todos los registros con el valor de la clave de búsqueda correspondiente.
- Los índices secundarios tienen que ser siempre densos.

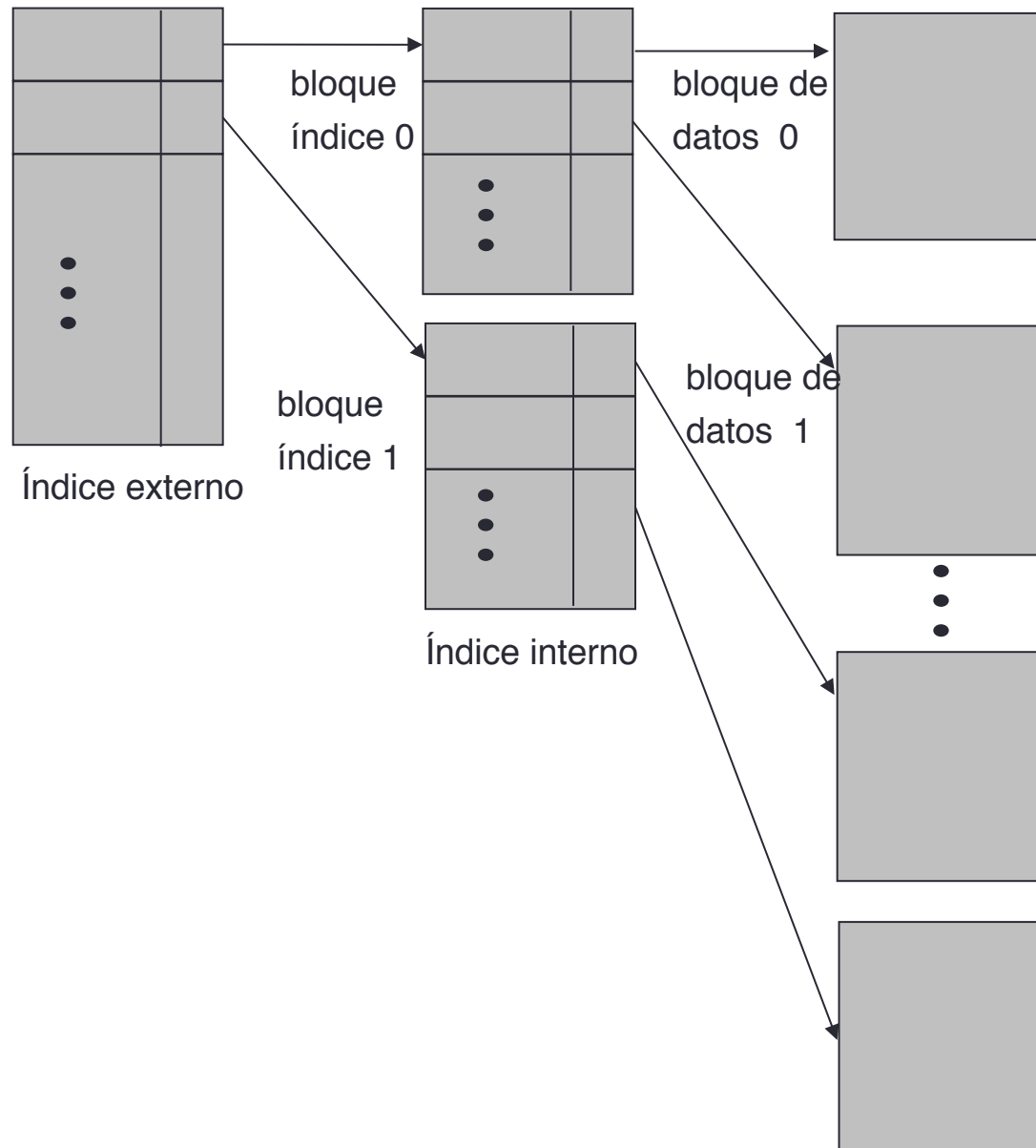
Índices primarios y secundarios

- Los índices ofrecen beneficios importantes en las búsquedas de registros.
- PERO: Cuando se modifica un fichero, se debe actualizar cada índice del fichero. La actualización de índices supone sobrecargas en las modificaciones de las bases de datos.
- Los accesos secuenciales utilizando índices primarios son eficientes, pero los accesos secuenciales utilizando índices secundarios son costosos.
 - Cada registro accedido puede suponer acceder a un nuevo bloque del disco.

Índices multinivel

- Si el índice primario no cabe en memoria, el acceso se hace costoso.
- Para reducir el número de accesos a disco a registros de índice, se trata de guardar el índice primario como un fichero secuencial y construir un índice disperso sobre él.
 - Índice externo – un índice disperso del índice primario
 - Índice interno – el fichero índice primario
- Si todavía el índice externo es demasiado grande para caber en memoria principal, se puede crear otro nivel de indexado, y así sucesivamente.
- Se deben actualizar los índices de todos los niveles cuando el fichero se actualice o se produzcan inserciones o borrados.

Índices multinivel (Cont.)



Modificación de índices: Borrado

- Si el registro borrado era el único registro del fichero con ese valor concreto de la clave de búsqueda, se borra también la clave de búsqueda del índice.
- Borrado en índice de nivel único:
 - Índice denso – el borrado de la clave de búsqueda es similar al borrado de un registro de un fichero.
 - Índice disperso – si existe en el índice una entrada para la clave de búsqueda, se borra reemplazando la entrada en el índice con el siguiente valor de la clave de búsqueda en el fichero (ordenado por la clave de búsqueda). Si el siguiente valor de la clave de búsqueda ya tiene una entrada en el índice, la entrada se borra en vez de reemplazarla.

Modificación de índices : Inserción

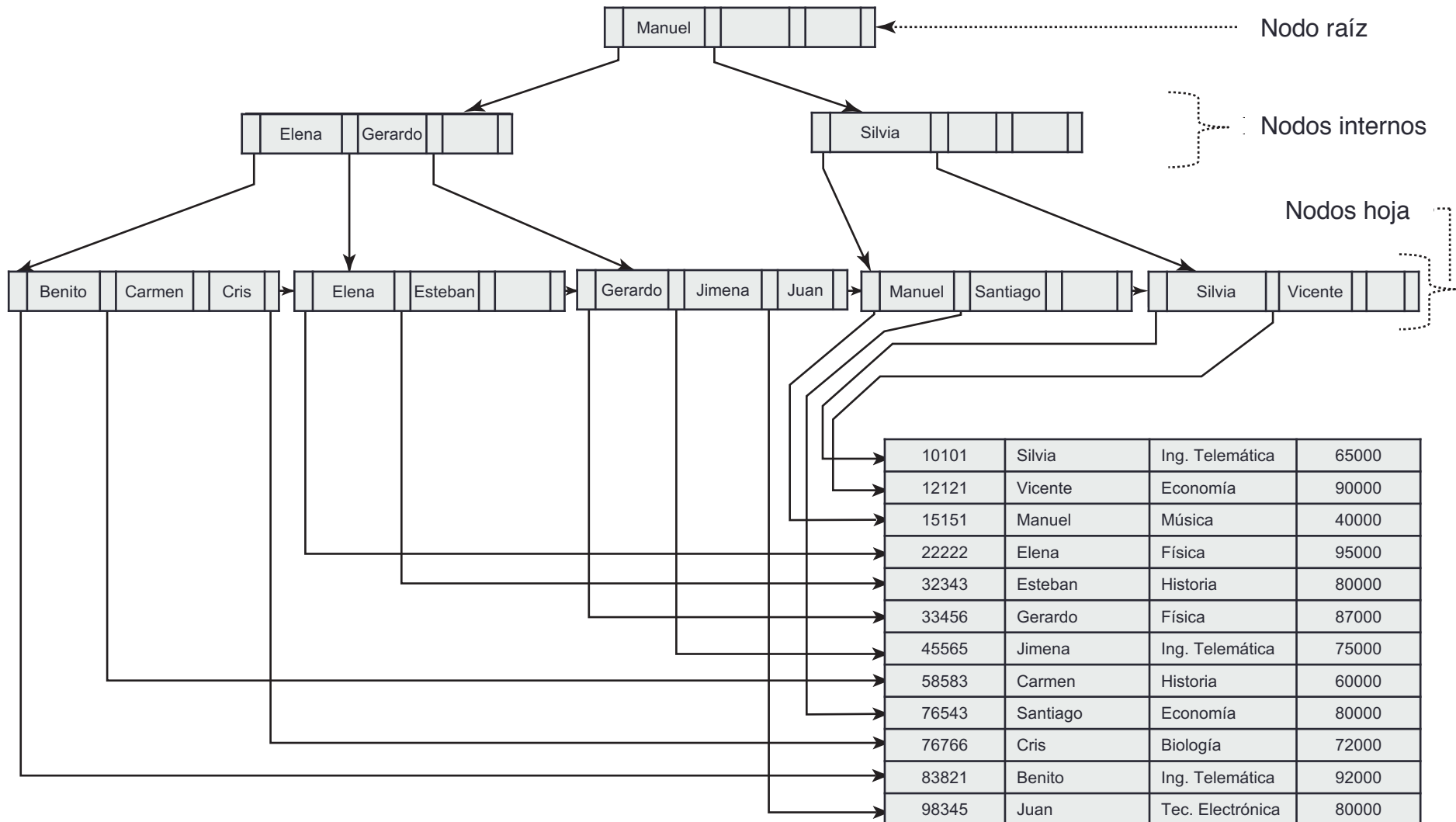
- Inserción en un índice de nivel único:
 - Se realiza una búsqueda utilizando el valor de la clave de búsqueda que aparece en el registro a insertar.
 - Índices densos – si el valor de la clave de búsqueda no aparece en el índice, se inserta.
 - Índices dispersos – si el índice almacena una entrada para cada bloque del fichero, no se necesitan hacer cambios en el índice a no ser que se cree un nuevo bloque. Si es así, el primer valor de la clave de búsqueda que aparezca en el nuevo bloque se inserta en el índice.
- Los algoritmos de inserción multinivel (y también borrado) son simples extensiones de los algoritmos de un único nivel

Ficheros índice tipo árbol B⁺

Los índices tipo árbol B⁺ son una alternativa a los ficheros índice secuenciales

- Desventaja de los ficheros índice secuenciales:
 - Se degradan las prestaciones cuando el fichero crece, dado que se tienen que crear muchos bloques de desbordamiento.
 - Es necesaria la reorganización periódica del fichero completo.
- Ventaja de los fichero índice tipo árbol B⁺:
 - Se reorganizan automáticamente ante cambios pequeños y locales como son inserciones y borrados.
 - No se necesita una reorganización del fichero completo para mantener las prestaciones.
- Desventaja (menor) de los árboles B⁺:
 - Sobrecarga adicional de inserciones y borrados, sobrecarga de espacio.
- Las ventajas de los árboles B⁺ son mayores que las desventajas:
 - Se utilizan con mucha frecuencia.

Ejemplo de un árbol B+



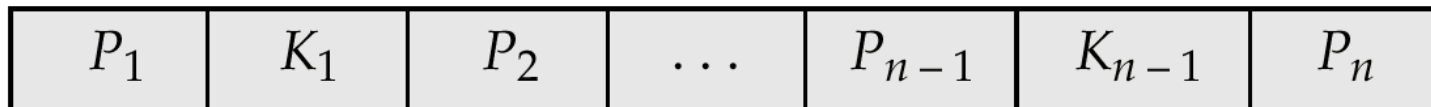
Ficheros índice tipo árbol B⁺ (Cont.)

Un árbol B⁺ es un árbol con raíz que cumple las siguientes propiedades:

- Todos los caminos de la raíz a las hojas tienen la misma longitud
- Cada nodo que no es raíz ni hoja tiene entre $\lceil n/2 \rceil$ y n hijos.
- Un nodo hoja tiene entre $\lceil (n-1)/2 \rceil$ y $n-1$ valores
- Casos especiales:
 - Si la raíz no es una hoja, tiene al menos 2 hijos.
 - Si la raíz es una hoja (es decir, no hay otros nodos en el árbol), puede tener entre 0 y $(n-1)$ valores.

Estructura de un nodo de un árbol B⁺

- Nodo típico



- K_i son los valores de la clave de búsqueda
- P_i son punteros a hijos (para nodos no hoja) o punteros a registros o conjuntos de registros (para nodos hoja).
- Las claves de búsqueda de un nodo están ordenadas

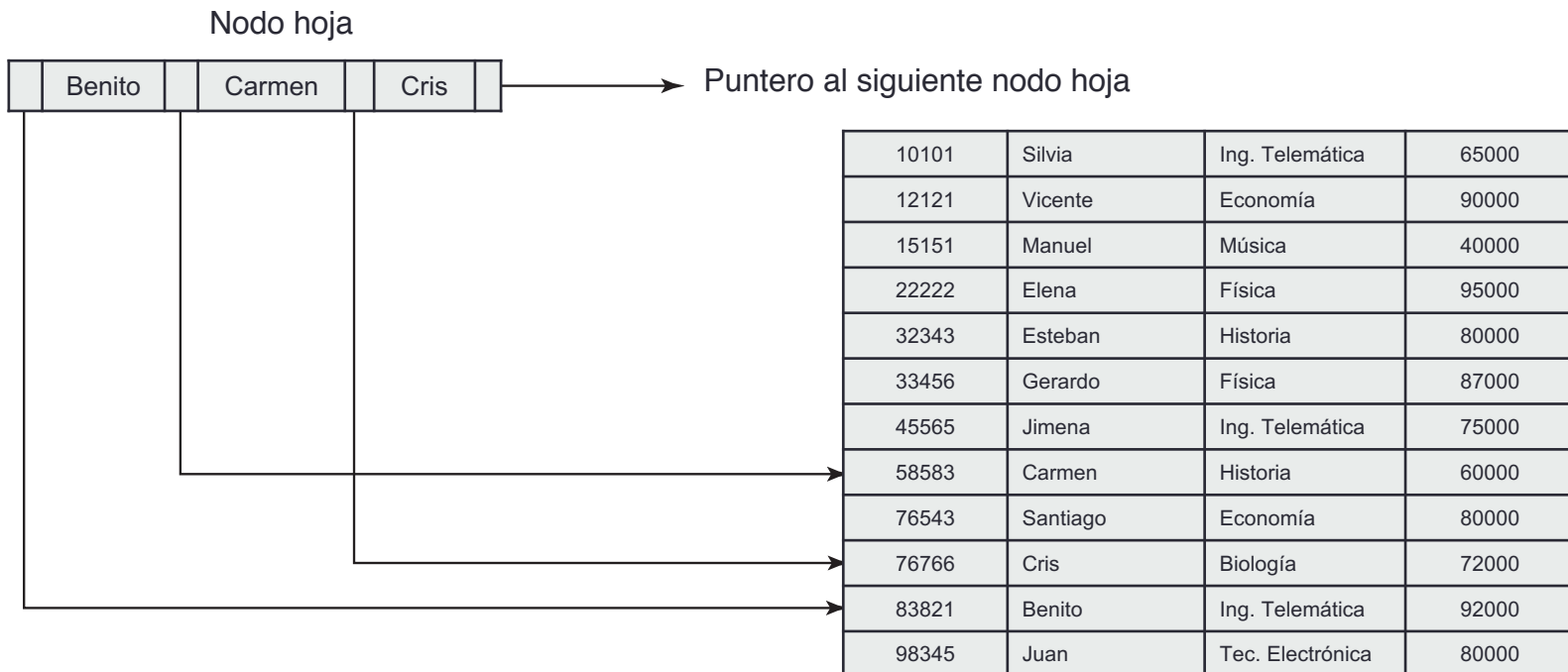
$$K_1 < K_2 < K_3 < \dots < K_{n-1}$$

(inicialmente vamos a suponer que no existen claves duplicadas)

Nodos hoja en árboles B⁺

Propiedades de un nodo hoja:

- Para $i = 1, 2, \dots, n-1$, el puntero P_i apunta a un registro de fichero con el valor de la clave de búsqueda K_i
- Si L_i, L_j son nodos hoja y $i < j$, los valores de la clave de búsqueda de L_i son menores que los valores de la clave de búsqueda de L_j .
- P_n apunta al siguiente nodo hoja en el orden de la clave de búsqueda.

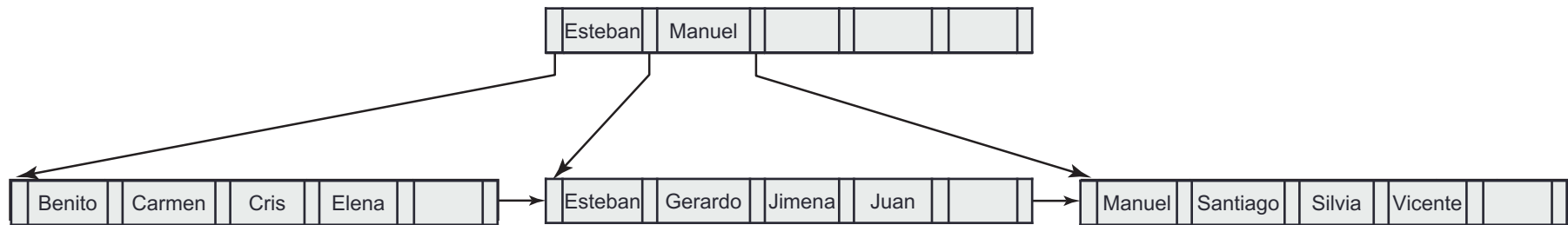


Nodos no hoja en árboles B⁺

- Los nodos no hoja forman un índice disperso multinivel para los nodos hoja. Para un nodo no hoja con m punteros:
 - Todas las claves de búsqueda en el subárbol a las que apunta P_1 son menores que K_1
 - Para $2 \leq i \leq n - 1$, todas las claves de búsqueda en el subárbol a las que apunta P_i tienen valores mayores o iguales que K_{i-1} y menores que K_i
 - Todas las claves de búsqueda en el subárbol al que apunta P_n tienen valores mayores o iguales que K_{n-1}

P_1	K_1	P_2	\dots	P_{n-1}	K_{n-1}	P_n
-------	-------	-------	---------	-----------	-----------	-------

Ejemplo de un árbol B+



Árbol B+ para el fichero *docente* ($n = 6$)

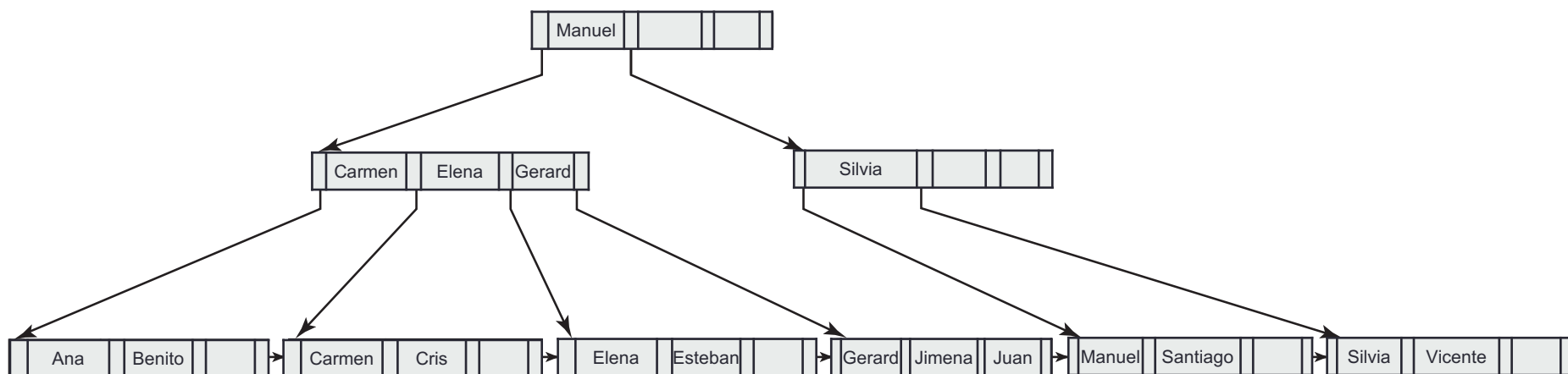
- Los nodos hoja deben tener entre 3 y 5 valores ($\lceil (n-1)/2 \rceil$ y $n-1$, con $n = 6$).
- Los nodos no hoja que no sean el raíz deben tener entre 3 y 6 hijos ($\lceil n/2 \rceil$ y n con $n = 6$).
- El nodo raíz debe tener al menos 2 hijos.

Observaciones sobre los árboles B⁺

- Dado que las conexiones entre nodos se realizan mediante puntero, los bloques “lógicamente” próximos no tienen por que estar “físicamente” próximos.
- Los niveles no hoja del árbol B⁺ forman una jerarquía de índices dispersos.
- El árbol B⁺ contiene un número relativamente pequeño de niveles (logarítmico respecto al tamaño del fichero principal), por lo que las búsquedas se pueden hacer eficientemente.
- Las inserciones y borrados en el fichero principal se pueden gestionar de manera eficiente, ya que el índice se puede reestructurar en tiempo logarítmico.

Consultas sobre árboles B⁺

- Encontrar el registro con un valor de la clave de búsqueda V .
 1. $C = \text{root}$
 2. WHILE C no sea nodo hoja {
 1. Sea i el menor valor tal que $V \leq K_i$
 2. IF no existe, hacer $C = \text{último puntero no nulo en } C$
 3. ELSE {IF ($V = K_i$) $C = P_{i+1}$ ELSE $C = P_i$ }}
 3. Sea i el menor valor que cumple $K_i = V$
 4. IF existe un valor para i , seguir el puntero P_i al nodo deseado
 5. ELSE no existe el registro buscado.



Gestionando duplicados

- Con duplicados en las claves de búsqueda:
 - Tanto en los nodos hoja como en los internos:
 - no podemos garantizar que $K_1 < K_2 < K_3 < \dots < K_{n-1}$
 - pero podemos garantizar que $K_1 \leq K_2 \leq K_3 \leq \dots \leq K_{n-1}$
 - Las claves de búsqueda en el subárbol al que apunta P_i :
 - son $\leq K_i$, pero no necesariamente $< K_i$
 - Veamos por qué: supongamos que el mismo valor de la clave de búsqueda V está en dos nodos hoja L_i y L_{i+1} . Entonces su nodo padre K_i debe ser igual a V .

Gestionando duplicados

- Podemos modificar el procedimiento de la siguiente forma:
 - atravesar P_i aún cuando $V = K_i$
 - Tan pronto como lleguemos a un nodo hoja C , comprobamos si C tiene solamente valores de la clave de búsqueda menores que V
 - si es así, hacemos $C =$ hermano derecho de C antes de comprobar si C contiene V
- Procedimiento para encontrar todas las ocurrencias
 - utilizamos el procedimiento modificado anterior para encontrar la primera ocurrencia de V
 - atravesamos las hojas consecutivas para encontrar todas las ocurrencias de V

Consultas sobre árboles B⁺ (Cont.)

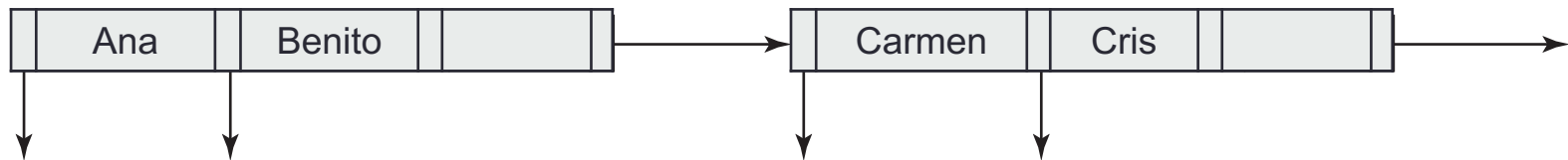
- En el procesamiento de una consulta, se sigue un camino en el árbol desde el nodo raíz a un nodo hoja.
- Si hay K valores de la clave de búsqueda en el fichero, el camino no es mayor de $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$
- Un nodo tiene normalmente el mismo tamaño que un bloque de disco, típicamente 4 kilobytes,
 - y n es normalmente de alrededor de 100 (40 bytes por entrada del índice)
- Con 1 millón de valores de la clave de búsqueda y $n = 100$:
 - se deben acceder en una búsqueda como mucho $\log_{50}(1,000,000) = 4$ nodos.
- Esto contrasta con un árbol binario balanceado con 1 millón de valores de la clave de búsqueda — se debe acceder a unos 20 nodos.
 - La diferencia anterior es significativa dado que cada acceso a un nodo puede necesitar hacer E/S al disco, lo que puede suponer unos 20 milisegundos!

Actualizaciones de árboles B⁺: Inserción

1. Encontrar el nodo hoja en el que debería aparecer el valor de la clave de búsqueda.
2. Si el valor de la clave de búsqueda ya está en un nodo hoja
 1. se añade el registro al fichero principal.
 2. si es necesario se añade un puntero en el “cajón”.
3. Si el valor de la clave de búsqueda no está,
 1. se añade el registro al fichero principal y se crea un “cajón” si es necesario.
 2. si hay espacio en el nodo hoja, se inserta un par (valor-clave, puntero) en el nodo hoja.
 3. si no, se divide el nodo (junto con la nueva entrada (valor-clave, puntero)) como se indica a continuación.

Actualizaciones de árboles B⁺: Inserción (Cont.)

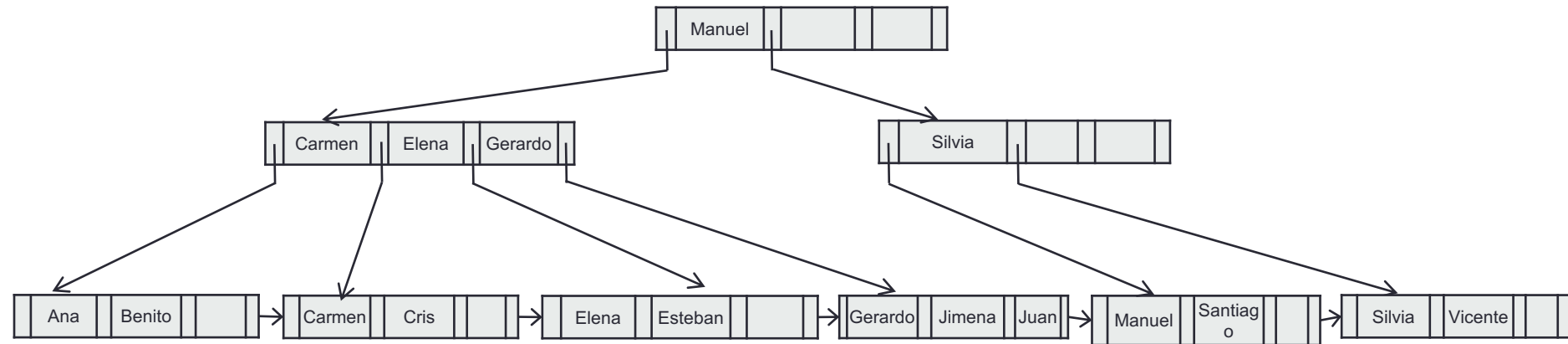
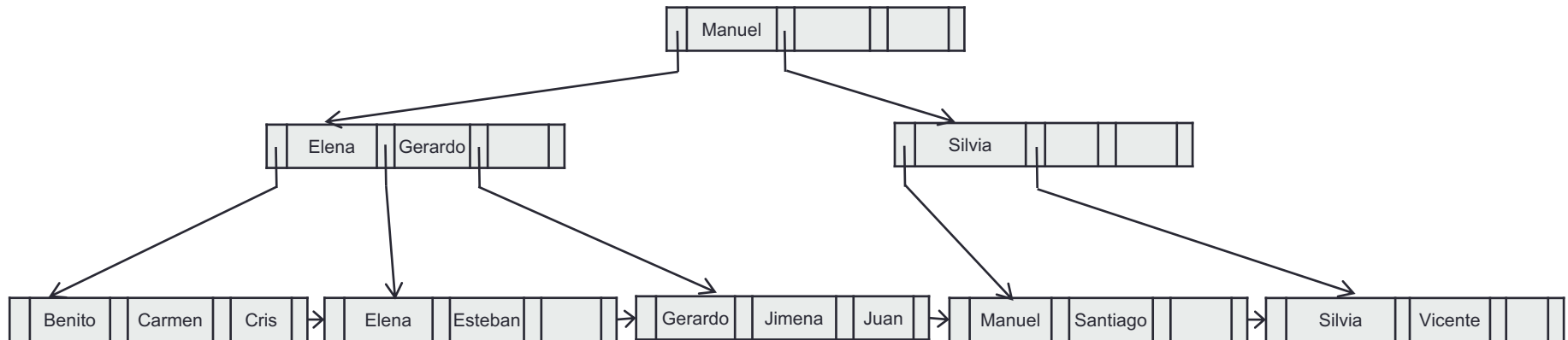
- Dividir un nodo hoja:
 - Coger los n pares (valor-clave, puntero) (incluyendo el que se va a insertar) ordenados. Situar los primeros $\lceil n/2 \rceil$ en el nodo original y el resto en un nuevo nodo.
 - Dado que el nuevo nodo sea p , y k sea el valor de la clave más bajo en p , insertar (k,p) en el padre del nodo que se está dividiendo.
 - Si el padre está lleno, dividirlo y **propagar** la división hacia arriba.
- La división de nodos se propaga hacia arriba hasta que se encuentra un nodo no lleno.
 - En el caso peor se debe dividir el nodo raíz aumentando en 1 la profundidad del árbol.



Resultado de dividir el nodo conteniendo Benito, Carmen y Cris con la inserción de Ana.

Siguiente paso: insertar la entrada con (Carmen, puntero al nuevo nodo) en el padre.

Actualizaciones de árboles B⁺: Inserción(Cont.)



Árbol B⁺ antes y después de insertar “Ana”

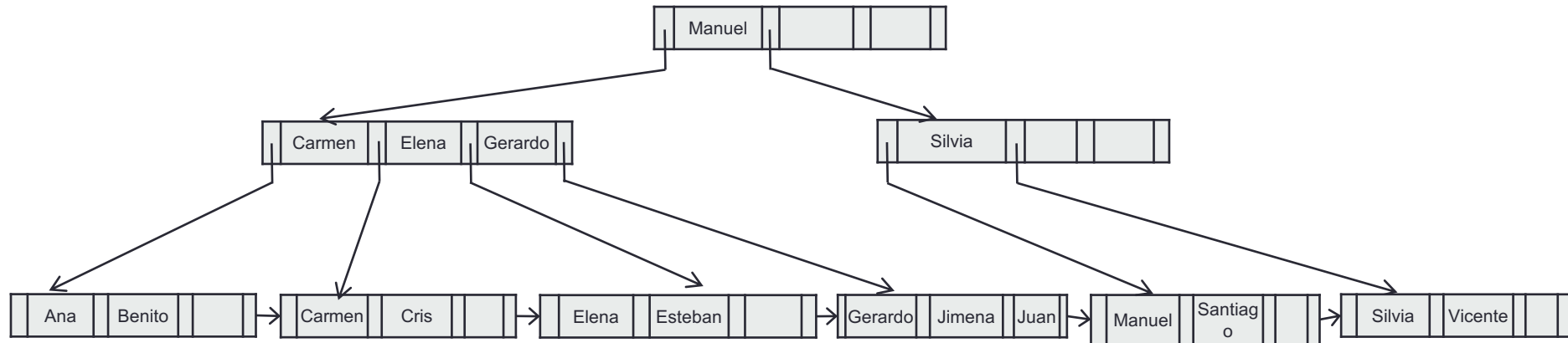
Actualización de árboles B⁺: Borrado

- Encontrar el registro a borrar y eliminarlo del fichero principal y del “cajón” (si existe)
- Eliminar (valor-clave, puntero) del nodo hoja si no hay “cajón” o si el “cajón” se vacía.
- Si el nodo se queda con un número insuficiente de entradas y la entradas del nodo y de un nodo hermano caben en un solo solo nodo, entonces
 - Insertar todos los valores de la clave en un solo nodo (el de la izquierda) y borrar el otro nodo.
 - Borrar el par (K_{i-1}, P_i) , donde P_i es el puntero al nodo borrado, desde su padre, recursivamente utilizando el procedimiento anterior.

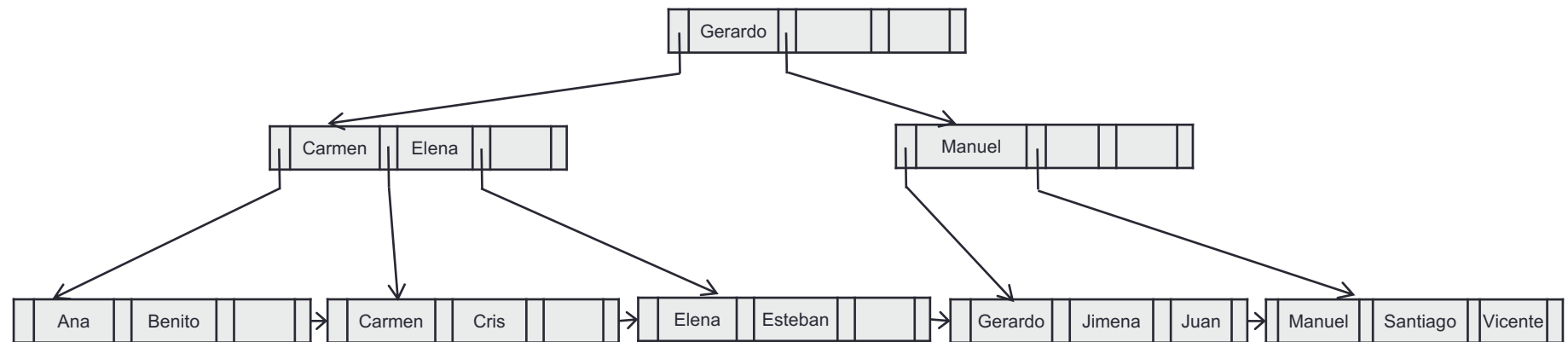
Actualización de árboles B⁺: Borrado

- Si no, si el nodo se queda con insuficientes entradas y las entradas en el nodo y un hermano no caben en un nodo, entonces
 - Redistribuir los punteros entre el nodo y un hermano tal que entre ambos tengan más del número mínimo de entradas.
 - Actualizar el valor correspondiente de la clave de búsqueda en el nodo padre.
- El borrado de nodos se puede propagar hacia arriba hasta encontrar un nodo que tenga $\lceil n/2 \rceil$ o más punteros.
- Si el nodo raíz solo tiene un puntero después del borrado, se borra y el único hijo pasa a ser el raíz.

Ejemplo de borrado en árbol B⁺

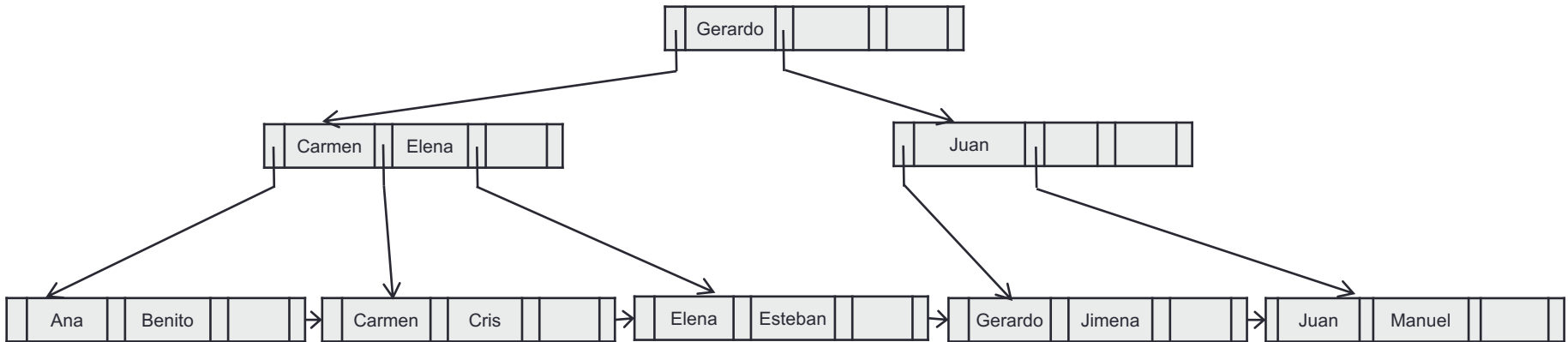


Antes y después de borrar “Silvia”



- El borrado de “Silvia” causa la mezcla de hojas con insuficientes nodos.

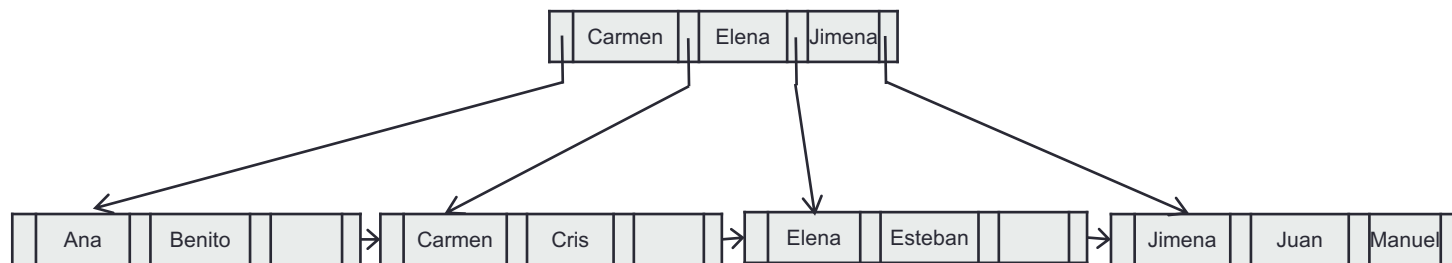
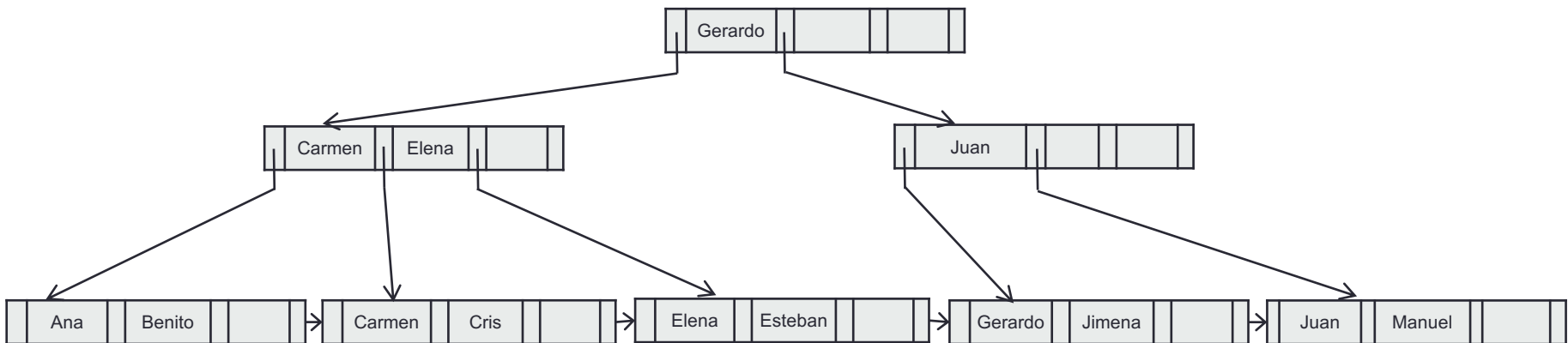
Ejemplo de borrado en árbol B⁺



Borrado de “Santiago” y “Vicente” del resultado de la operación anterior.

- La hoja que contiene Santiago y Vicente queda sin suficientes nodos, y “coge prestado” un valor Juan de su hermano izquierdo.
- El valor de la clave de búsqueda en el padre cambia como resultado.

Ejemplo de borrado en árbol B⁺



Borrado de “Gerardo” del resultado del ejemplo anterior

- El nodo con Gerardo y Jimena queda con insuficientes nodos y se junta con su hermano.
- Como resultado el nodo padre queda con insuficientes nodos y se junta con su hermano (y se borra una entrada de su padre)
- El nodo raíz que ‘pasa a tener un solo hijo, y se borra y su hijo pasa a ser la nueva raíz.

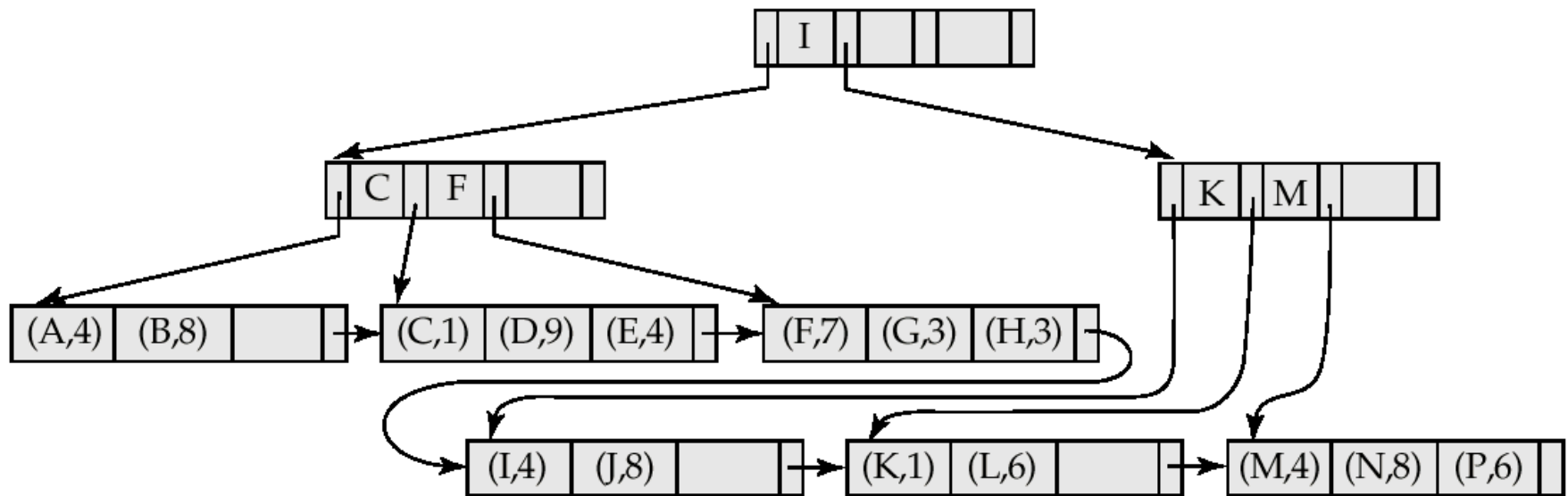
Claves de búsqueda no únicas

- Alternativas al esquema descrito:
 - “Cajones” de punteros en bloques separados (mala idea)
 - Listas de punteros a tuplas con cada clave
 - Necesario código extra para manejar listas.
 - Borrar una tupla puede ser costoso si hay muchos duplicados de la clave de búsqueda
 - Poca necesidad de espacio adicional, no tiene coste extra para las búsquedas
 - Hacer la clave de búsqueda única añadiendo un identificador de registro
 - Se necesita espacio adicional para las claves
 - Código más simple para insertar/borrar
 - Muy utilizado.

Organización de ficheros tipo árbol B⁺

- El problema de la degradación de ficheros índice se resuelve utilizando índices tipo árbol B⁺.
- El problema de la degradación de ficheros de datos se resuelve utilizando una organización de ficheros tipo árbol B⁺.
- Los nodos hoja en una organización de ficheros de tipo árbol B⁺ almacenan registros en vez de punteros.
- Los nodos hoja siguen teniendo que estar medio llenos.
 - Dado que los registros son más grandes que los punteros, el número máximo de registros que se pueden almacenar en un nodo hoja es menor que el número de punteros en un nodo no hoja.
- Las inserciones y borrados se tratan de la misma forma que las inserciones y borrados de entradas en un índice de tipo árbol B⁺.

Organización de ficheros tipo árbol B⁺ (Cont.)



Ejemplo de organización de fichero de tipo árbol B⁺

- Es importante una buena utilización del espacio ya que los registros utilizan más espacio que los punteros.
- Para mejorar la utilización de espacio, utilizar más hermanos en la redistribución durante las divisiones y uniones
 - Utilizar 2 hermanos en las redistribuciones (para evitar dividir/juntar si es posible) resulta en que cada nodo tiene al menos $\lfloor 2n/3 \rfloor$ entradas

Ficheros índice de tipo B

- Similares a los árboles B^+ , pero en los árboles B los valores de la clave de búsqueda sólo pueden aparecer una vez; se elimina el almacenamiento redundante de claves de búsqueda.
- Las claves de búsqueda en los nodos no hoja no aparecen en ningún otro sitio en el árbol B; se debe incluir un campo puntero adicional para cada clave de búsqueda en un nodo no hoja.
- Nodo hoja de un árbol B generalizado



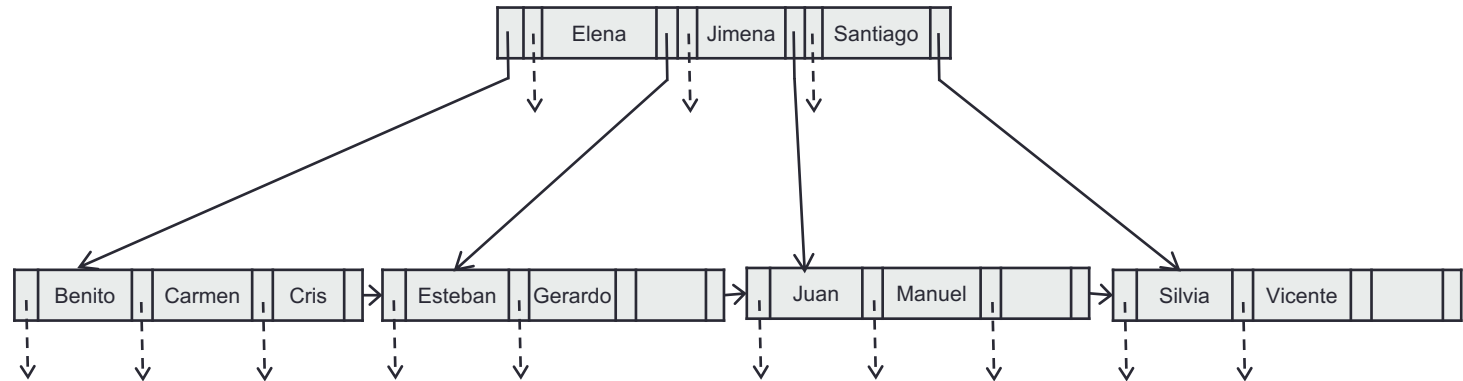
(a)



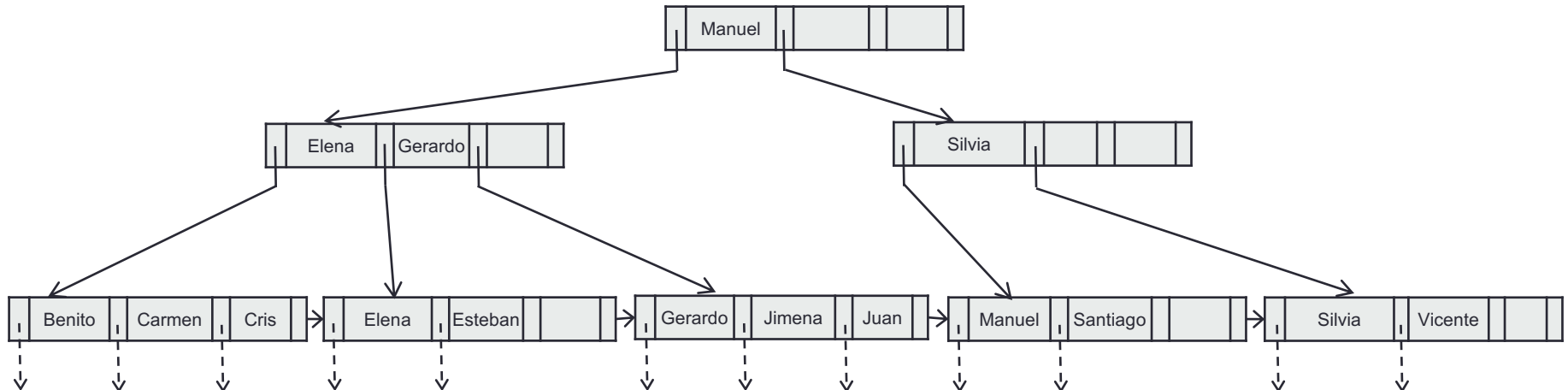
(b)

- Nodo no hoja – los punteros B_i son los conjuntos de punteros registros del fichero.

Ejemplo de ficheros índice de tipo B



El árbol B (arriba) y el árbol B+ (abajo) contienen los mismos datos



Ficheros índice de tipo B (Cont.)

- Ventajas de los índices de tipo árbol B:
 - Pueden utilizar menos nodos que el árbol B^+ equivalente.
 - A veces se puede encontrar el valor de la clave de búsqueda sin llegar a un nodo hoja.
- Desventajas de los índices de tipo árbol B:
 - Solo una pequeña fracción de todos los valores de la clave de búsqueda se encuentran antes
 - Los nodos no hoja son más grandes, por lo que caben menos claves. Por ello, los árboles B suelen tener más profundidad que los árboles B^+ equivalentes.
 - Las inserciones y borrados son más complejos que en árboles B^+ .
 - La implementación es más compleja que la de los árboles B^+ .
- Normalmente, las ventajas de los árboles B no compensan sus desventajas.

Asociación (hash) estática

- Un **cajón** (**bucket**) es una unidad de almacenamiento que contiene uno o más registros (un cajón típicamente es un bloque de disco).
- En una **organización de archivos asociativa** se obtiene el cajón de un registro directamente desde su valor de clave de búsqueda, empleando una función de asociación.
- La función de asociación h es una función desde el conjunto de todos los valores de claves de búsqueda K , hasta el conjunto de todas las direcciones de cajones B .
- La función de asociación se emplea para localizar registros para accesos, inserciones y borrados.
- Los registros con diferentes valores de claves de búsqueda pueden asociarse al mismo cajón; de este modo, para localizar un registro, se ha de recorrer secuencialmente el cajón entero.

Ejemplo de organización de archivos asociativa

La organización de archivos asociativa del archivo *docente*, utilizando *nombre-dpto* como clave (véase la figura de la siguiente transparencia).

- Hay 10 cajones,
- La representación binaria del carácter *i-ésimo* se asume que sea el entero *i*.
- La función de asociación devuelve la suma de las representaciones binarias de los caracteres módulo 10
 - P.e. $h(\text{Música}) = 1$ $h(\text{Historia}) = 2$ $h(\text{Física}) = 3$
 $h(\text{Tecnología Electrónica}) = 3$

Ejemplo de organización de archivos asociativa

Cajón 0

Cajón 1

15151	Manuel	Música	40000

Cajón 2

32343	Esteban	Historia	80000
58583	Carmen	Historia	60000

Cajón 3

22222	Elena	Física	95000
33456	Gerardo	Física	87000
98345	Juan	Tec. Ele	80000

Cajón 4

12121	Vicente	Econom	90000
76543	Santiago	Econom	80000

Cajón 5

76766	Cris	Biología	72000

Cajón 6

10101	Silvia	Ing. Tel.	65000
45565	Jimena	Ing. Tel.	75000
83821	Benito	Ing. Tel.	92000

Cajón 7

Organización de archivos asociativa del archivo *docentes* utilizando *nombre-dpto* como clave.

Funciones de asociación

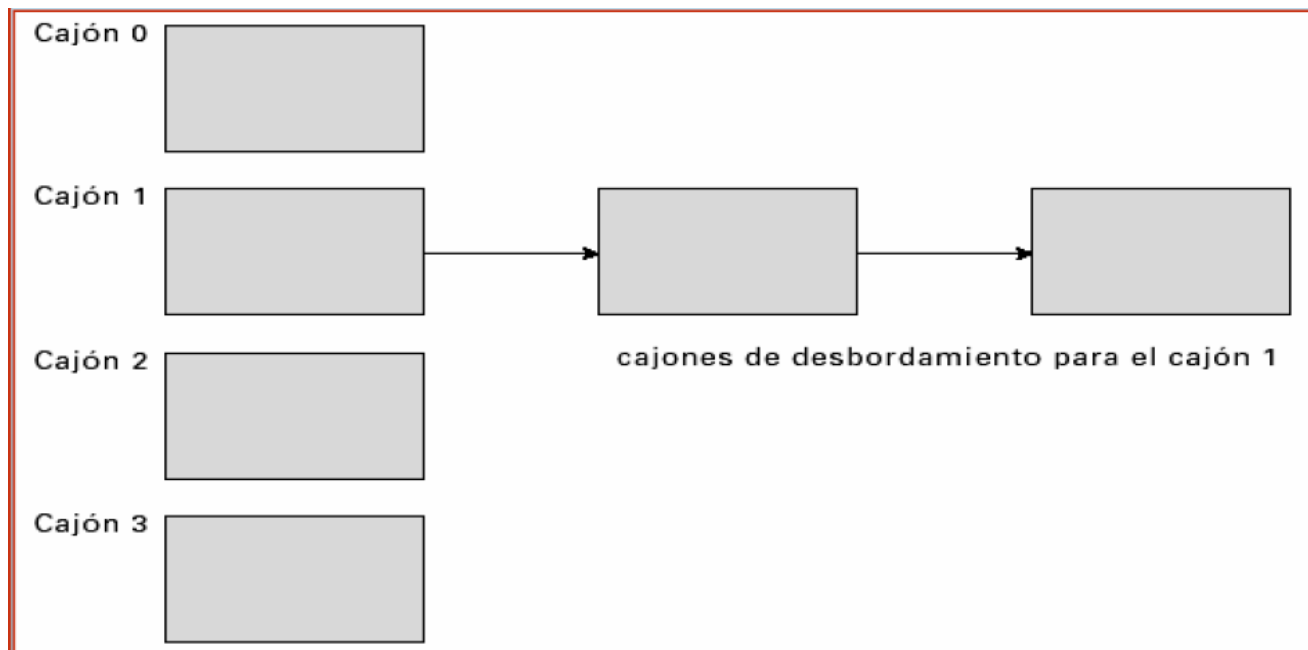
- La peor función de asociación asocia todos los valores de las claves de búsqueda al mismo cajón; esto hace que el tiempo de acceso sea proporcional al número de valores de claves de búsqueda en el archivo.
- Una función de asociación ideal es **uniforme**, es decir, cada cajón se asigna al mismo número de valores de claves de búsqueda, desde el conjunto de todos los valores posibles.
- La función de asociación ideal es **aleatoria**; así cada cajón tendrá asignado el mismo número de registros, independientemente de la distribución real de los valores de las claves de búsqueda en el archivo.
- Las funciones de asociación típicas realizan cálculos sobre la representación binaria interna de la clave de búsqueda.
 - Por ejemplo, para una clave de búsqueda de secuencia de caracteres, se podrían añadir las representaciones binarias de todos los caracteres en la secuencia y se podría devolver la suma del número de cajones.

Gestión de desbordamiento de cajones

- El desbordamiento de cajones puede producirse por
 - Insuficientes cajones
 - Desviación en la distribución de los registros. Esto puede tener lugar por dos razones:
 - múltiples registros tienen el mismo valor de clave de búsqueda
 - la función de asociación elegida produce una distribución no uniforme de los valores de las claves
- Aunque se puede reducir la probabilidad de desbordamiento de cajones, no se puede eliminar y se gestiona empleando *cajones de desbordamiento*.

Gestión de desbordamiento de cajones

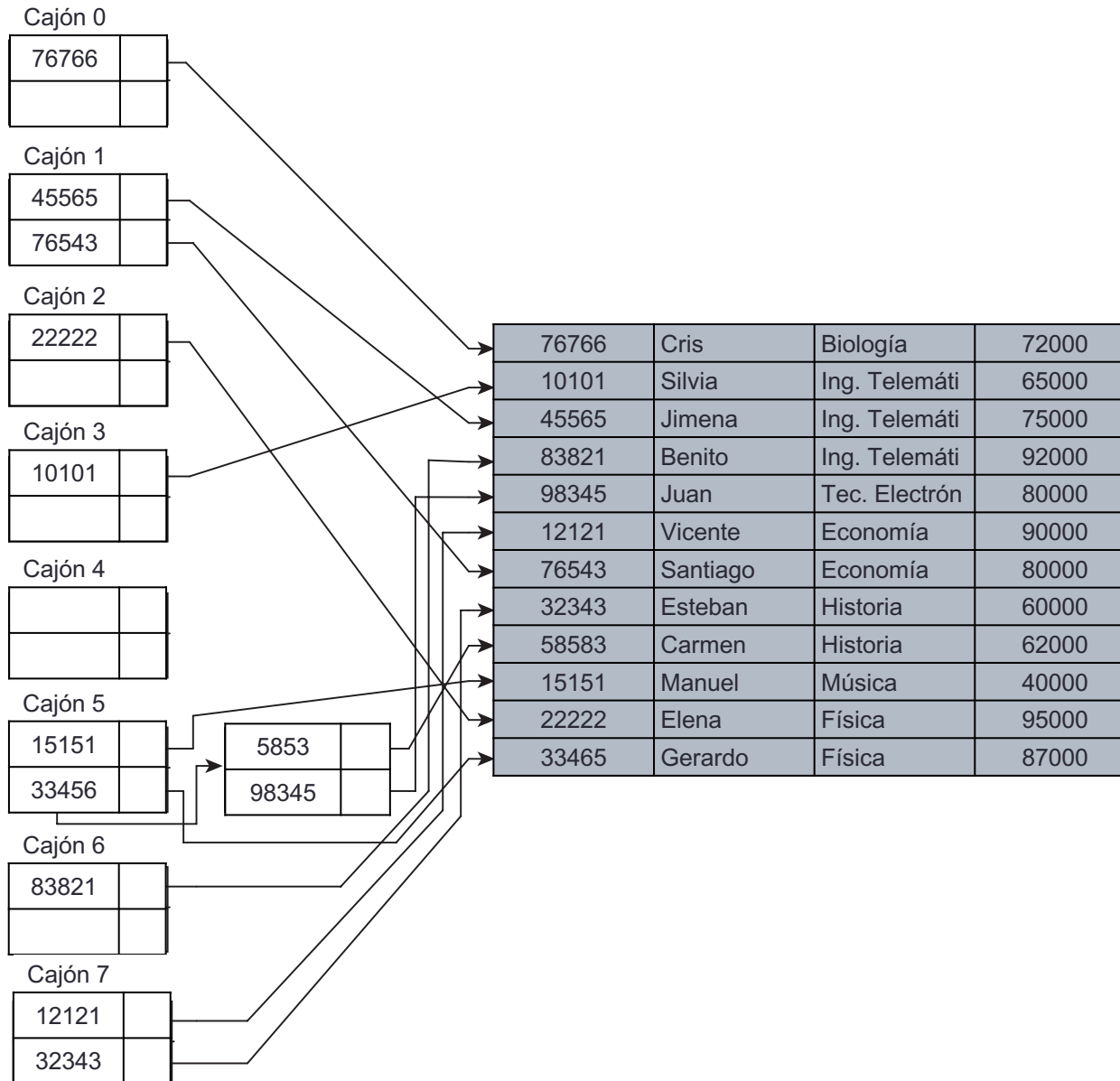
- **Cadena de desbordamiento** –los cajones de desbordamiento de un determinado cajón se encadenan juntos en una lista enlazada.
- El esquema anterior se denomina **asociación cerrada**.
 - Una alternativa, denominada **asociación abierta**, que no emplea cajones de desbordamiento, no es adecuada para aplicaciones de bases de datos.



Índices asociativos

- La asociación no sólo se puede emplear en la organización de archivos, sino también para la creación de estructuras de índices.
- Un **índice asociativo** organiza las claves de búsqueda, con sus punteros de registros asociados, en una estructura de archivos asociativa.
- En su sentido estricto, los índices asociativos son siempre índices secundarios
 - si el propio archivo está organizado empleando asociación, no es necesario un índice asociativo primario independiente de él, que emplee la misma clave de búsqueda.
 - Sin embargo, se emplea el término índice asociativo para referirse, tanto a las estructuras de índices secundarios, como a los archivos organizados en asociación.

Ejemplo de índice asociativo



Deficiencias de la asociación estática

- En la asociación estática la función h asocia valores de claves de búsqueda a un determinado conjunto **fijo** B , de direcciones de cajones. Y las bases de datos crecen con el tiempo.
 - Si el número inicial de cajones es demasiado pequeño, disminuirá el rendimiento debido a los muchos desbordamientos.
 - Si se anticipa el tamaño del archivo en algún momento del futuro, y en consecuencia el número de cajones asignados, un aumento significativo de espacio se desperdiciará inicialmente.
 - Si disminuye la base de datos, nuevamente se desperdiciará espacio.
- Una opción es la reorganización periódica del archivo con una nueva función de asociación, pero
 - es muy costoso.
 - interrumpe las operaciones normales.
- Estos problemas se pueden evitar empleando técnicas que permitan que el número de cajones se modifique dinámicamente.



Asociaciones dinámicas

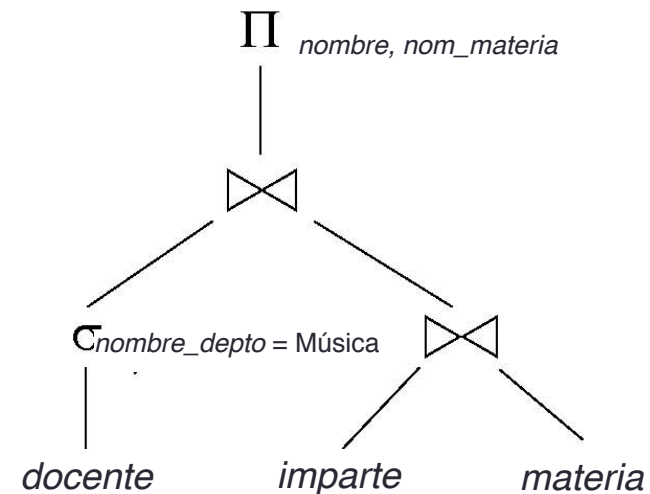
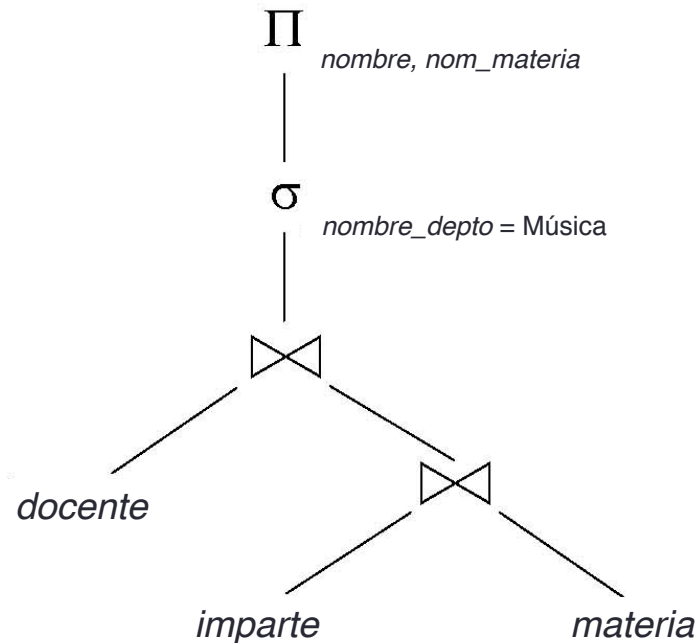
Comparación de índices y asociaciones

- Coste de reorganización periódica.
- Frecuencia relativa de inserciones y eliminaciones.
- ¿Es deseable optimizar el tiempo de acceso medio a costa del tiempo de acceso en caso peor)
- Tipos de consultas esperadas:
 - Las asociaciones normalmente son mejores obteniendo registros que tengan un valor determinado de la clave.
 - Si son habituales las consultas por rangos, los índices son mejores.
- En la práctica:
 - PostgreSQL soporta índices asociativos, pero desaconseja su uso ya que tienen muy malas prestaciones.
 - Oracle soporta organización con asociaciones estáticas, pero no índices asociativos.
 - SQLServer e InnoDB (MySQL) sólo soportan árboles B⁺

OPTIMIZACIÓN DE CONSULTAS

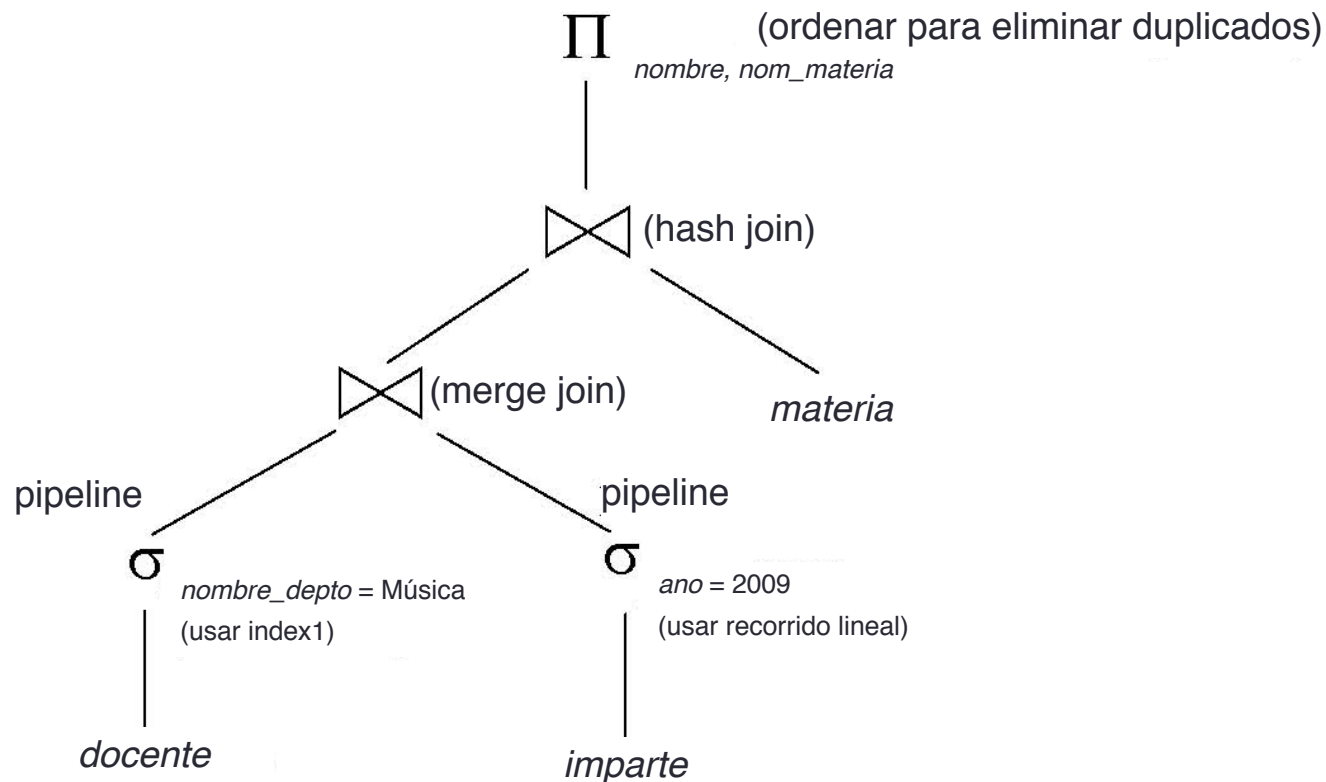
Introducción

- Formas alternativas de evaluar una consulta dada
 - Expresiones equivalentes
 - Diferentes algoritmos para cada operación



Introducción (Cont.)

- Un **plan de evaluación** define de forma exacta que algoritmos se utilizan para cada operación, y cómo se coordina la ejecución de las operaciones.



Introducción (Cont.)

- La diferencia de coste entre planes de evaluación de una consulta puede ser enorme
 - P.e. segundos vs. días, en algunos casos.
- Pasos de la **optimización de consultas basada en coste**
 1. Generar expresiones lógicamente equivalente utilizando **reglas de equivalencia**
 2. Anotar las expresiones resultantes para disponer de planes de consulta alternativos.
 3. Elegir el plan más “barato” en base al **coste estimado**.
- La estimación del coste de un plan se basa en:
 - Información estadística sobre las relaciones. Por ejemplo:
 - número de tuplas, número de valores distintos de un atributo, ...
 - Estimaciones estadísticas para resultados intermedios
 - para calcular el coste de expresiones complejas.
 - Cálculos de coste de los algoritmos, calculados utilizando estadísticas.

Generando expresiones equivalentes

- Dos expresiones de álgebra relacional se dice que son equivalentes si las dos generan el mismo conjunto de tuplas para cada instancia posible de la base de datos.
 - El orden de las tuplas es irrelevante.
- Las reglas de equivalencia indican cuándo dos expresiones con forma diferente son equivalentes
 - se pueden reemplazar expresiones con la forma de la primera por la segunda y viceversa.

Ejemplo de reglas de equivalencia

1. Las operaciones de selección conjuntivas se pueden descomponer en una secuencia de selecciones individuales.

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. La operación de selección es conmutativa.

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

3. Sólo es necesaria la última de una secuencia de proyecciones, las demás se pueden omitir.

$$\Pi_{L_1}(\Pi_{L_2}(\dots(\Pi_{L_n}(E))\dots)) = \Pi_{L_1}(E)$$

4. Las operaciones de join natural son asociativas:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

5. ...

Ejemplo de regla de optimización

- Para todas las relaciones r_1, r_2 , y r_3 ,

$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$

(Asociativa de join natural)

- Si $r_2 \bowtie r_3$ es grande y $r_1 \bowtie r_2$ es pequeña, elegimos

$$(r_1 \bowtie r_2) \bowtie r_3$$

de tal forma que hay que calcular y almacenar una relación temporal más pequeña.

Enumeración de expresiones equivalentes

- Los optimizadores de consultas utilizan reglas de equivalencia para generar expresiones equivalente de forma **sistemática** para una expresión dada
- Pueden generar todas las expresiones equivalentes de la siguiente forma:
 - REPEAT
 - aplicar todas las reglas de equivalencia aplicables a cada subexpresión de cada expresión equivalente encontrada.
 - añadir las expresiones generadas al conjunto de expresiones equivalentes.
 - UNTIL no se generen más expresiones equivalentes
- Es una aproximación muy costosa en espacio y tiempo
 - Dos aproximaciones alternativas:
 - Generación de planes optimizados en base a reglas de transformación.
 - Aproximación especial para consultas con sólo selecciones, proyecciones y joins

Elección de planes de evaluación

- Cuando se eligen planes de evaluación se deben considerar las interacciones de las técnicas de evaluación
 - elegir el algoritmo más barato para cada operación independientemente puede no llevar al mejor algoritmo global.
- Los optimizadores de consulta en la práctica incorporan elementos de dos grandes tipos aproximaciones:
 1. Buscar todos los planes y elegir el mejor en base a costes.
 2. Utilizar heurísticos para elegir el plan.

Optimización basada en costes

- Consideremos encontrar el mejor orden de hacer los joins en $r_1 \bowtie r_2 \bowtie \dots r_n$.
- Hay $(2(n-1))!/(n-1)!$ órdenes de hacer los joins diferentes para la expresión.
 - Con $n = 7$, el número es de 665.280.
 - Con $n = 10$, el número es mayor de 176.000.000.000
- No necesitamos generar todas las ordenaciones. Utilizando técnicas de optimización como la programación dinámica, obtenemos que el orden de menor coste para cualquier subconjunto de $\{r_1, r_2, \dots r_n\}$ se puede calcular una sola vez y almacenarlo para futuros usos.

Coste de la optimización

- Con técnicas de programación dinámica, la complejidad temporal de la optimización de un conjunto de joins es $O(3^n)$.
 - Con $n = 10$, este número es 59.000 frente a 176.000.000.000
- La complejidad espacial es $O(2^n)$
- Con posteriores optimizaciones se puede llegar a complejidad temporales en encontrar el mejor orden de un conjunto de joins de $O(n 2^n)$
 - La complejidad espacial se mantiene en $O(2^n)$
- La optimización basada en costes es costosa, pero no habitual, para consultas de grandes conjuntos de datos (las consultas típicas tienen un n bajo, normalmente < 10)

Optimización heurística

- La optimización basada en costes es costosa, aún utilizando técnicas de programación dinámica.
- Los optimizadores pueden usar *heurísticos* para reducir el número de opciones que se deben analizar con estimación de costes.
- La optimización heurística transforma el árbol de consulta utilizando un conjunto de reglas que típicamente (aunque no siempre) mejoran las prestaciones de ejecución:
 - Realizar selecciones pronto (reducen el número de tuplas)
 - Realizar proyecciones pronto (reducen el número de atributos)
 - Realizar las selecciones y joins más restrictivos (es decir, con tamaño de resultado más pequeño) antes que otras operaciones similares.
 - Algunos sistemas utilizan solamente heurísticos, otros combinan heurísticos con optimización parcial basada en costes.

FIN DEL TEMA 4
