

Temario

- ◆ Introducción y fundamentos
- ◆ Introducción a SQL
- ◆ Modelo Entidad / Relación
- ◆ Modelo relacional
- ◆ Diseño relacional: formas normales
- ◆ Consultas
 - Cálculo relacional
 - Álgebra relacional
- ◆ Implementación de bases de datos
 - Estructura física: campos y registros
 - Indexación
 - Índices simples
 - Árboles B
 - Hashing
 - Compresión

Implementación de un SGBD

- ◆ Después de SQL, modelo E/R, modelo relacional...
 - Modelo físico de datos: la misma información en **estructuras de archivos** (bytes)
Relaciones y atributos → “bytes” en disco
- ◆ Implementación de las operaciones relacionales sobre datos almacenados en disco
 - Consultas
 - Actualización: inserción, modificación, eliminación
- ◆ Optimizaciones para hacer viables las operaciones de consulta y actualización
 - Análisis del coste de las operaciones en disco
 - Estrategias de borrado/inserción/compactación
 - **Indexación**: índices simples, árboles B, hashing
 - Compresión

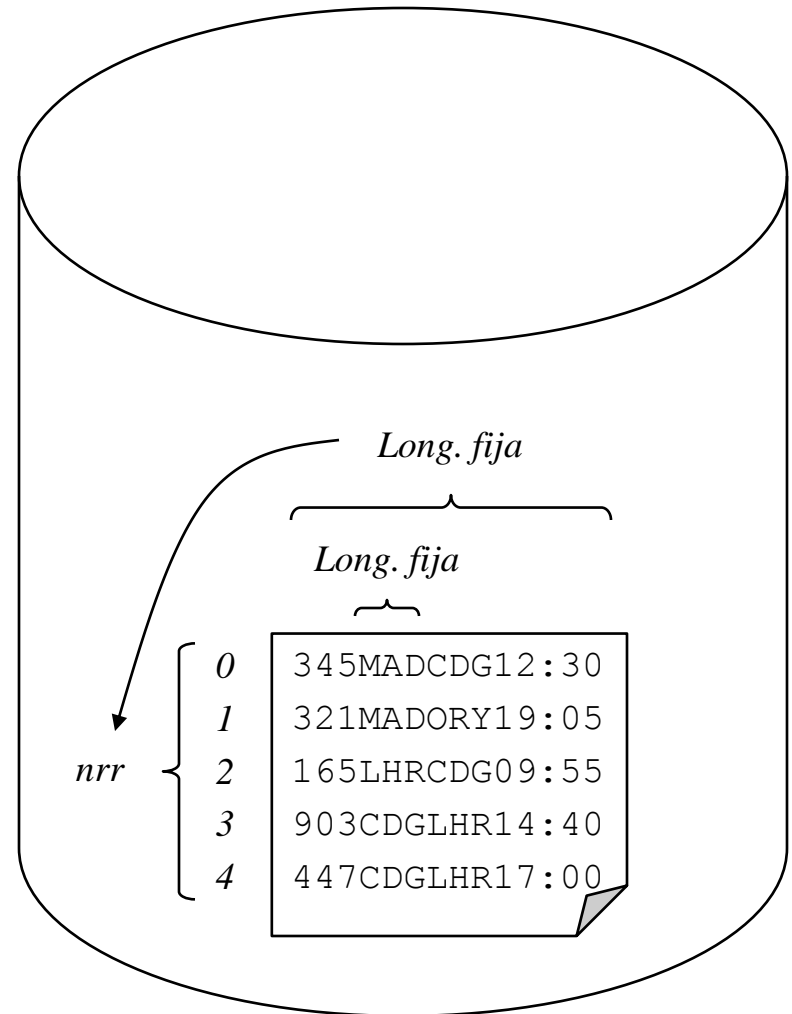
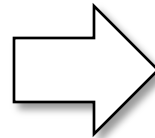
Temario

- ◆ Introducción y fundamentos
- ◆ Introducción a SQL
- ◆ Modelo Entidad / Relación
- ◆ Modelo relacional
- ◆ Diseño relacional: formas normales
- ◆ Consultas
 - Cálculo relacional
 - Álgebra relacional
- ◆ Implementación de bases de datos
 - Estructura física: campos y registros
 - Indexación
 - Índices simples
 - Árboles B
 - Hashing
 - Compresión

Almacenamiento de tablas: Campos y registros

- ◆ Tupla → registro, atributo → campo
- ◆ Delimitación de campos y registros
 - Longitud fija
 - Longitud variable: separador o indicador de longitud
- ◆ Cabeceras con “autodescripción” de estructuras, nº de registros, etc.

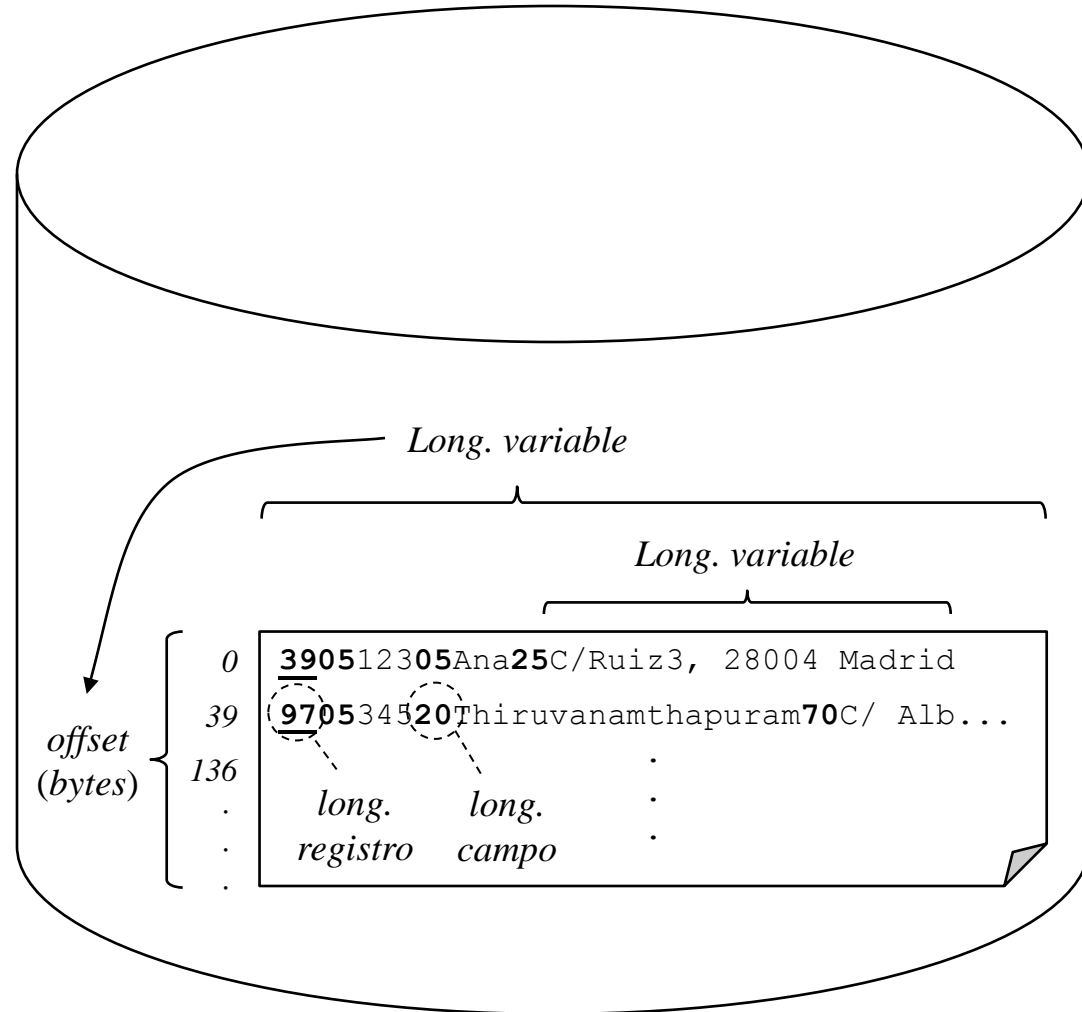
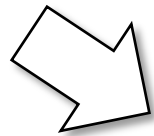
Numero	Origen	Destino	Salida
345	MAD	CDG	12:30
321	MAD	ORY	19:05
165	LHR	CDG	09:55
903	CDG	LHR	14:40
447	CDG	LHR	17:00



Almacenamiento de tablas: Campos y registros

EMPLEADO

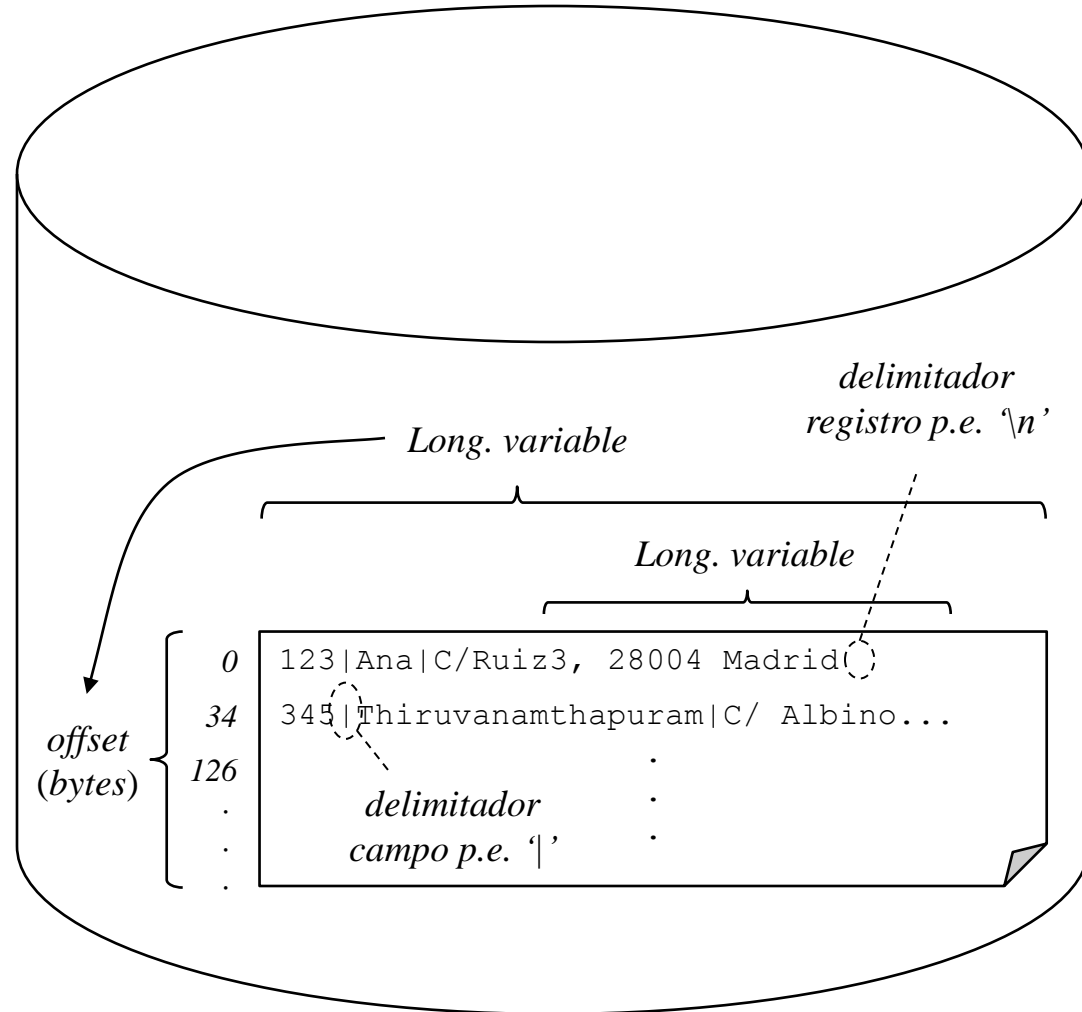
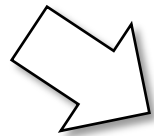
DNI	Nombre	Dirección
123	Ana	C/ Ruiz 3, 28004 Madrid
345	Thiruvanamthapuram	C/ Albino Pérez Ayestarain 14, San Fernando de Henares, 28830 Madrid
⋮	⋮	⋮
⋮	⋮	⋮



Almacenamiento de tablas: Campos y registros

EMPLEADO

DNI	Nombre	Dirección
123	Ana	C/ Ruiz 3, 28004 Madrid
345	Thiruvanamthapuram	C/ Albino Pérez Ayestarain 14, San Fernando de Henares, 28830 Madrid
⋮	⋮	⋮
⋮	⋮	⋮



Delimitación de registros y campos

- ◆ Longitud fija
 - Acceso a datos más rápido
 - Mayor gasto de espacio en disco
 - Los registros se direccionan por nº de orden (nrr)
- ◆ Longitud variable
 - Acceso más lento
 - Optimización de espacio en disco
 - Puede interesar cuando la variabilidad de tamaño de registros es muy amplia (p.e. campos muy variables y/u opcionales)
 - Los registros se direccionan por su posición en bytes (offset)

Operaciones en disco

- ◆ Acceso
 - Acceso a un registro por posición
 - Búsqueda de registros por condiciones (consultas)
 - Índices para optimizar las operaciones de acceso
- ◆ Actualización
 - Inserción
 - Modificación, eliminación: típicamente conllevan una consulta
 - Estrategias de borrado eficiente combinado con inserción
- ◆ Coste de acceso a datos en disco
 - Es un aspecto clave y condiciona todas las técnicas
 - En disco es varios órdenes de magnitud mayor al acceso en RAM
 - Formulaciones sencillas de operaciones comunes resultan inviables
 - Cambia completamente el análisis de complejidad
 - Cambian las estrategias y algoritmos con respecto a las técnicas en RAM

Costes de acceso

- ◆ Estructura de una unidad de disco magnético
 - Discos, superficies
 - Cilindros, pistas, sectores
 - Clusters y bloques
- ◆ Operaciones de lectura/escritura y su coste
 - Seek
 - Latencia rotacional
 - Transferencia

Estrategias de eliminación / inserción

- ◆ Eliminación: marcar el registro, dejando un hueco
- ◆ Formar una lista con los huecos disponibles
- ◆ Periódicamente, compactar
- ◆ Al insertar, buscar un hueco (en otro caso, al final del archivo)
→ se retrasa la necesidad de compactación
- ◆ Con registros de longitud fija, la estrategia es sencilla
- ◆ Con longitud variable: first-fit, best-fit, worst-fit

Temario

- ◆ Introducción y fundamentos
- ◆ Introducción a SQL
- ◆ Modelo Entidad / Relación
- ◆ Modelo relacional
- ◆ Diseño relacional: formas normales
- ◆ Consultas
 - Cálculo relacional
 - Álgebra relacional
- ◆ Implementación de bases de datos
 - Estructura física: campos y registros
 - Indexación
 - Índices simples
 - Árboles B
 - Hashing
 - Compresión

Ejecución de consultas

- ◆ Parsing de SQL a estructura (árbol) de consulta (álgebra) en RAM
- ◆ Reducción de la consulta a unión/intersección o conjunción/disyunción de condiciones simples de comparación de atributos
- ◆ Las condiciones de comparación atributo / constante se pueden resolver por búsqueda binaria en un índice
- ◆ Las condiciones de comparación entre dos atributos se resuelven por procesamiento cosecuencial
- ◆ Las operaciones de agregación se pueden aplicar después (el agrupamiento añade alguna complejidad extra)

Consultas simples

- ◆ Comparación atributo / constante –para simplificar consideremos $t . A = cte$
- ◆ Recorrer todos los registros hasta encontrar los que cumplen la condición?
 - $O(n)$ con $n = n^\circ$ de registros en la tabla
- ◆ Ordenar (y mantener ordenado) el archivo de datos por el atributo A en cuestión, y hacer búsqueda binaria?
 - $O(\log n)$, pero el mantener el orden con cada actualización es $O(n)$, no es viable
 - Además sólo resuelve consultas de un sólo atributo
 - Incluso, se puede conseguir mejor que $O(\log n)$?
- ◆ Solución óptima: índices!
 - Búsqueda en $O(1)$
 - Sin limitación en los campos de búsqueda

Indexación

- ◆ En términos abstractos, un índice es un mapping φ_A
 - $\varphi_A : \text{valor de } A \rightarrow \text{dirección de los registros con ese valor en el campo } A$
- ◆ Se necesita un índice aparte para cada campo que se quiera buscar
- ◆ La “ejecución” del mapping debe ser muy poco costosa, idealmente $O(1)$
- ◆ Dada la dirección del registro (o registros) que con el valor buscado, sólo queda hacer 1 acceso por registro (obviamente, menos es imposible)
- ◆ Técnicas principales de indexación para BDs
 - Índices simples: el mapping lo da un array de pares valor campo / dirección en RAM
 - Árboles B: semejantes a los índices simples, pero en vez de un array, la estructura es más sofisticada (un árbol con técnicas de actualización elaboradas)
 - Hashing: el mapping lo da una función que se utiliza tanto para posicionar los registros al insertar, como al buscarlos por campo

Índices simples

- ◆ Cabe en RAM y las operaciones en él (búsqueda, actualización, reordenación, etc.) tienen coste mínimo comparado con el acceso a disco
- ◆ Ordenado para permitir búsqueda binaria y búsquedas por desigualdad (rangos)
- ◆ Permite búsqueda por tantos campos como se desee, sin tocar el fichero de datos (en realidad cualquier método de indexación)

Índices simples (cont)

- ◆ Un índice simple es un array de pares *valor campo / identificador registro*
 - Es común llamar “clave” al valor de campo
 - No confundir con las nociones de claves en el modelo relacional, E/R, SQL
 - El “identificador” de registro puede ser una dirección física (identificador físico) o una clave primaria (identificador lógico)
- ◆ Distinción básica
 - Índices primarios: indexan una clave primaria (y usan direcciones de registros)
 - Índices secundarios: todos los demás (usan claves primarias como id de registros)
- ◆ El índice primario juega un papel especial: es el que realmente contiene direcciones de registros

Ejemplo...

	<u>ID</u>	<i>Director</i>	<i>Título</i>	<i>Fecha</i>
0	062622	Kubrick, Stanley	2001 Odisea del espacio	1968
49	054331	Kubrick, Stanley	Espartaco	1960
83	050825	Kubrick, Stanley	Senderos de Gloria	1957
125	047478	Kurosawa, Akira	Los siete samurais	1954
157	042876	Kurosawa, Akira	Rashomon	1950
192	071411	Kurosawa, Akira	Dersu Uzala	1975
225	110413	Besson, Luc	Leon	1994
249	758043	Vadim, Jean	Leon	1992

Operaciones – índice primario

Operaciones en RAM Operaciones en disco
--

- ◆ Crear índice
 - Leer fichero de datos y construir array en RAM
 - Ordenar array
 - Guardar array en archivo
- ◆ Cargar índice en RAM (si es posible)
 - Lectura (secuencial por bloques) del archivo de índice → índice en RAM
- ◆ Búsqueda de un registro
 - Búsqueda (binaria) en índice → dirección (offset) del registro buscado
 - Acceso (seek + read) al registro sabiendo su posición

Operaciones – índice primario (cont)

Operaciones en RAM Operaciones en disco
--

- ◆ Inserción de registro
 - Archivo de datos: escribir el registro en la dirección d que corresponda
 - Índice: añadir entrada $\langle clave, d \rangle$ manteniendo el orden
- ◆ Eliminación de registro
 - Archivo de datos: eliminar por el método de gestión del espacio que se desee
 - Índice: eliminar entrada correspondiente
 - Compactación \rightarrow reconstruir el índice
- ◆ Modificación de un campo
 - Archivo de datos: modificar el valor del campo
 - Índice: si el campo es parte de la clave primaria: eliminar + insertar en el índice
 - Se mueve el registro \rightarrow actualizar su dirección en el índice (o eliminar + insertar)

Operaciones – índice secundario

Operaciones en RAM Operaciones en disco
--

- ◆ Búsqueda de registro
 - Buscar clave secundaria en índice secundario → clave primaria
 - Buscar en índice primario → dirección del registro en archivo de datos
 - Acceder al registro en disco, leer
- ◆ Inserción de registro
 - Archivo de datos: insertar el registro
 - Índice: Igual que índice primario
- ◆ Eliminación de registro
 - Archivo de datos: eliminar el registro
 - Índice: se puede posponer la eliminación
 - El intento de acceso a un registro borrado se detecta a nivel del índice primario
 - Periódicamente, limpiar el índice secundario

Operaciones – índice primario (cont)

Operaciones en RAM Operaciones en disco
--

- ◆ Modificación de un campo
 - Archivo de datos: modificar
 - Índice: si afecta a la clave secundaria, recolocar para mantener el orden
 - Si afecta a la clave primaria, actualizar en el secundario la clave cambiada
 - Si la clave primaria aparece en entradas clave secundaria repetida, recolocar para mantener el orden de clave primaria
 - Si afecta a otros campos: el índice secundario no cambia

Uso de índices en consultas más generales

$\sigma_{A=c}$ (<i>tabla</i>)	Obvio con índice por campo A	$O(\log n)$
$\sigma_{A < c}$ (<i>tabla</i>)	Buscar c en índice A y tomar todas las anteriores	$O(n)$
$\sigma_{A=c \text{ and } B=d}$ (<i>tabla</i>)	Buscar c en índice $A \cap$ buscar d en índice B	$O(n)$
$\sigma_{A=c \text{ or } B=d}$ (<i>tabla</i>)	Buscar c en índice $A \cup$ buscar d en índice B	$O(n)$
$\sigma_{A=B}$ (<i>tabla</i>)	Índice A “ \cap ” índice B	$O(n)$
$\sigma_{A < B}$ (<i>tabla</i>), $\sigma_{A \text{ like } B}$ (<i>tabla</i>)	Comparaciones “todos con todos”	$O(n^2)$
$T \bowtie_{A,B} S$	Índice A de $R \cap$ índice B de S + combinar registros	$O(n)$
...		

Obsérvese que...

- ♦ La ordenación de los índices es necesaria para obtener esta complejidad en las operaciones
 - Aún en los peores casos ($n^{\text{nº campos}}$), son operaciones en RAM
- ♦ El resultado de las operaciones en índices es un conjunto de direcciones de registros
 - A partir de ahí, el coste de acceso a los registros es inevitable, inherente a la consulta
 - El uso de índices no implica acceso a disco en ningún momento (salvo guardar y cargar índices)
 - En consultas que implican varias tablas, el resultado puede ser un conjunto de *tuplas* de registros

Límite de los índices simples

- ◆ Los índices simples son una solución óptima para la ejecución de consultas
 - Imposible mejorar 0 accesos a disco! ☺
 - Sólo si el archivo de datos entero cabe en RAM... y prescindimos de indexar
- ◆ Todo lo razonado hasta aquí presupone que los índices se manejan en RAM

¿Y si un índice no cabe en RAM...?

Temario

- ◆ Introducción y fundamentos
- ◆ Introducción a SQL
- ◆ Modelo Entidad / Relación
- ◆ Modelo relacional
- ◆ Diseño relacional: formas normales
- ◆ Consultas
 - Cálculo relacional
 - Álgebra relacional
- ◆ Implementación de bases de datos
 - Estructura física: campos y registros
 - Indexación
 - Índices simples
 - Árboles B
 - Hashing
 - Compresión

Árboles B

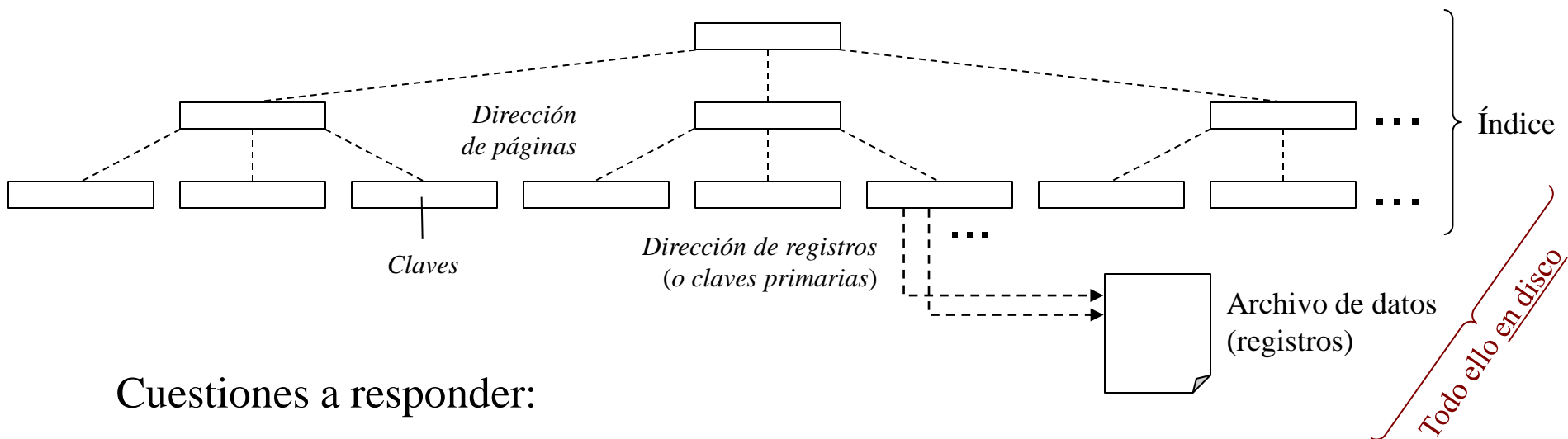
- ◆ Cuando un índice simple no cabe en RAM dejan de funcionar todos los supuestos que estábamos haciendo y la solución ya no es válida
- ◆ Con un índice en disco...
 - Buscar claves implica $\sim \log_2 n$ accesos a disco, mejor que $O(n)$ pero...
¿es posible algo mejor?
 - La actualización del índice supone $O(n)$ accesos a disco, ya no es viable

Árboles B (cont)

Ideas...

1. Dividir el índice en bloques que caben en RAM
 - Llamaremos “**páginas**” a estos bloques
 - Su tamaño puede ser p.e. el de bloque de disco
 - Interesa que las páginas se puedan leer secuencialmente de disco, sin saltos
2. Crear un índice de bloques
 - ¿Y si este índice no cabe en RAM?
3. Crear un índice de índices de bloques...
 - Finalmente tenemos un **índice multinivel** → estructura de **árbol**
 - Podemos utilizar el mismo tamaño de página en todos los niveles; así podemos almacenar todo en un único archivo homogéneo

Árboles B (cont)



Cuestiones a responder:

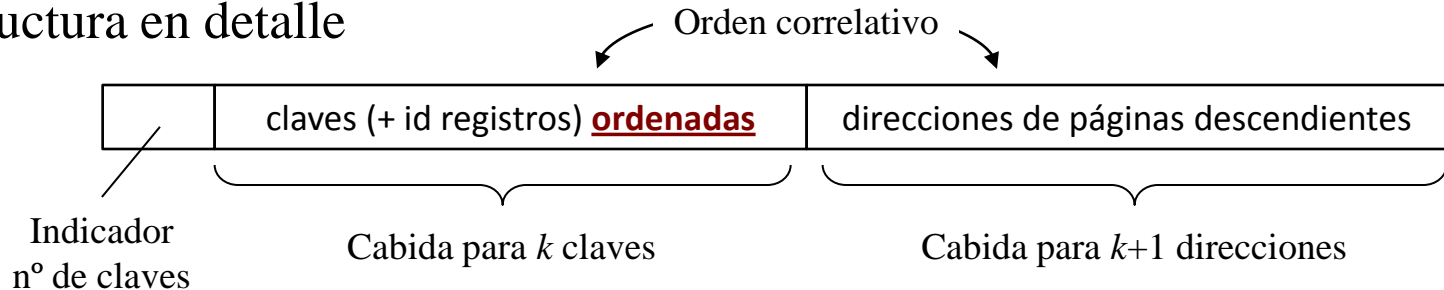
- ◆ ¿Qué **estructura** damos a las páginas? Claves, direcciones de páginas, de registros...
- ◆ ¿Debemos mantener algún **orden** dentro de las páginas y entre ellas?
- ◆ ¿Cómo **actualizamos** un índice de este tipo?

Árboles B → soluciones a estas preguntas: estructuras y algoritmos

Árboles B – Estructura

- ◆ Contenido de una página
 - Claves ordenadas acompañadas por dirección de página hija; cada clave “separa” dos páginas hijas
 - Las claves de cada página son “mayores” que las de la página que la “precede”
 - La clave que separa dos páginas es mayor que las claves de una y menor que las claves de la otra
 - Las direcciones de página en las páginas hoja son “null” (p.e. -1)

- ◆ Estructura en detalle



- ◆ La cabecera del archivo de índice debe contener la dirección de la página raíz
- ◆ Dos opciones en el almacenamiento de claves
 - Árboles B básicos: todas las páginas contienen claves del índice e id de registros
 - Árboles B+: las hojas contienen todas las claves, las demás contienen “separadores”

Ejemplo...

Árboles B – Operaciones

◆ **Búsqueda**

1. Iniciar en $np \leftarrow$ raíz
2. Si $np = -1$, **FIN** (no se encuentra)
3. Leer la página np a RAM, buscar la clave en la página
4. Si se encuentra la clave, **FIN** (clave encontrada)
5. Si no, $np \leftarrow$ página descendiente correspondiente al orden de la que se busca, volver al paso 2

◆ **Inserción**

1. Buscar la página hoja donde debería insertarse la clave (control de duplicados si es un índice primario)
2. Si hay espacio en la página para una clave más, insertar, **FIN**
3. Si no lo hay, dividir la página en dos, dejando una clave + n° página entre medias
4. Si la página era la raíz, crear una nueva raíz (aumenta la altura del árbol), **FIN**
5. Si no, subir la clave intermedia + n° de página a la página “padre”, volver al paso 2

Por este procedimiento, en general las páginas **no estarán completamente llenas**

- ◆ **Eliminación:** lo vemos en un momento... primero veamos ejemplos y hablemos de costes

Ejemplo...

Árboles B – Costes

- ◆ El coste de la búsqueda y de la inserción es... $O(h)$, con h = altura del árbol
- ◆ La altura del árbol es mayor cuanto menos llenas estén las páginas
- ◆ Si k es el máximo nº de claves por página, m es un mínimo que aseguremos, y n es el nº total de claves en el índice, se puede ver que la altura del árbol es...

$$1 + \log_k (n + 1) \leq h \leq 2 + \log_{m+1} [(n + 1) / 2]$$

- ◆ Por tanto, a mayor m , menor coste en las operaciones
- ◆ Con la técnica básica de inserción, aseguramos... $m = \lfloor k/2 \rfloor$
- ◆ ¿Podemos conseguir mejores mínimos? Lo veremos, p.e. árboles B*

Árboles B – Operaciones (cont)

◆ Eliminación

1. Buscar la página se encuentra la clave
2. Si no es una página hoja, intercambiar la clave con la clave “inmediatamente anterior” o posterior
3. Eliminar la clave de la página
4. Si la página tiene el mínimo de claves que queremos garantizar, **FIN**
5. En otro caso:
 - a) Intentar redistribuir con una página “hermana” inmediatamente contigua (en tal caso, **FIN**)
 - b) Si esto no es posible, concatenar con una página “hermana” inmediatamente contigua, sacando la clave que las separa en la página padreVolver al paso 3 con la página eliminada en la página padre

Si la concatenación se propaga hasta la raíz, se reduce un nivel en altura del árbol

Ejemplo...

Optimizaciones

- ◆ Árboles B virtuales
 - Se mantienen m páginas en RAM
 - Cuando se necesita una página que ya está en RAM, se evita su coste de lectura
 - Cuando se necesita una página que no está en RAM, ocupa el lugar de otra en RAM
 - Estrategias para decidir de qué página se prescinde: la más baja en el árbol, la usada menos recientemente, estadísticas de acceso...
- ◆ Árboles B*
 - Inserción con redistribución
 - Dividir dos páginas en tres
 - La ocupación mínima de las páginas es... $m = \lfloor 2k/3 \rfloor$
 - Asegurar este mínimo también al eliminar (en particular, concatenar “tres a dos”)
- ◆ Páginas de longitud variable
 - Cabrán más claves por página
 - El orden del árbol ya no es fijo –la capacidad de las páginas no se define en nº de claves, sino en bytes

Árboles B+

- ◆ Las claves (junto con los ids –dirección física o clave primaria– de registros) están todas en las hojas
 - No precisan n°s de página hijas, sólo referencia a registros
- ◆ Las páginas internas contienen separadores
 - No contienen referencia a registros
 - Los separadores pueden ser claves (p.e. la 1ª de la página hoja correspondiente) o prefijos de clave (de longitud mínima para discriminar)
- ◆ Ventaja de los árboles B
 - Permiten recorrido secuencial (búsquedas de rangos) de claves más eficiente que los árboles B básicos
 - Para ello, las páginas hoja deben enlazarse cada una con la siguiente: así se pueden recorrer sin subir a las páginas internas

Árboles B+ – Operaciones

- ◆ **Búsqueda**
 - Similar a los árboles B, pero no se detiene hasta llegar a las hojas
- ◆ **Inserción**
 - Cuando se divide una hoja, no se deja una clave en medio, sino que se crea un nuevo separador que se inserta en el siguiente nivel del árbol
- ◆ **Eliminación**
 - Cuando se redistribuyen claves en las páginas hoja, se substituye un separador por otro nuevo en la página padre
 - Cuando se concatenan páginas, se elimina un separador de la página padre (pero no se incluye en la página hoja)