



Universidad de Granada

decsai.ugr.es

Fundamentos de Bases de Datos

Grado en Ingeniería Informática

Tema 5: Nivel interno



DECSAI

**Departamento de Ciencias de la
Computación e Inteligencia Artificial**

1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



Una BD sirve para

Almacenar de forma permanente grandes cantidades de datos

Con el propósito principal de

Gestionar de forma eficiente los datos y su almacenamiento

Esto tiene sus consecuencias tanto en la organización lógica de los datos, como en su organización física



Nivel Interno

- ❑ Expresa en última instancia, las **operaciones** sobre los **datos** (creación, alteración y recuperación) en **términos** de actuación sobre unidades mínimas de almacenamiento denominadas **páginas o bloques de base de datos**.
- ❑ Provee al administrador de mecanismos para **optimizar** el **almacenamiento** y el **acceso a los datos**.
- ❑ Se encuentra **implementado** en el **SGBD**.

Nivel Físico

- ❑ Se encuentra **implementado** en el **Sistema Operativo**.
- ❑ Proporciona al SGBD una **capa** de **abstracción sobre** el **hardware**.
- ❑ Realiza el **acceso** a los medios de **almacenamiento** mediante **llamadas** a los **servicios** del sistema de archivos **proporcionado por el SO**.

1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



Características

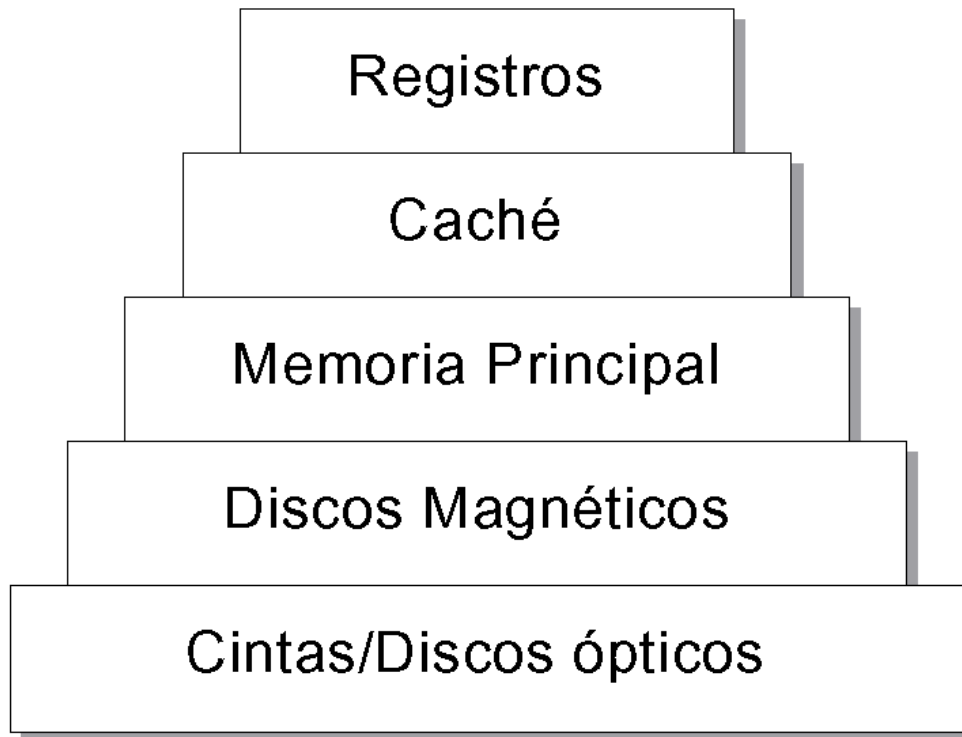
Almacenamiento
Primario


Almacenamiento
Secundario

Tipo Almacenamiento	Velocidad Acceso (<i>ms</i>)	Coste (\$/ <i>Mb</i>)	Capacidad (<i>Mb</i>)	Características.
Caché	10^{-6}	10	0-10	Volátil
Memoria Principal	10^{-5}	0.1	$0 - 10^4$	Volátil
Memoria Flash	10^{-4}	0.5-1	$0 - 10^4$	No Volátil
Discos magnéticos	5-10	10^{-3}	$10^4 - 10^5$	No volátil Acceso directo
Disquetes	> 150	0.1	$1 - 10^2$	No volátil
Discos ópticos	80-150	$.5 \cdot 10^{-3}$	$10^2 - 10^4$	Acceso directo No volátil
Cintas magnéticas	> 1000	10^{-3}	$10^2 - 10^6$	No volátil Acceso secuencial

Tabla comparativa de los distintos medios de almacenamiento

Jerarquía



- 
- menor coste
 - mayor capacidad
 - mayor tiempo acceso

Memoria principal

Es el dispositivo de almacenamiento primario de los ordenadores.

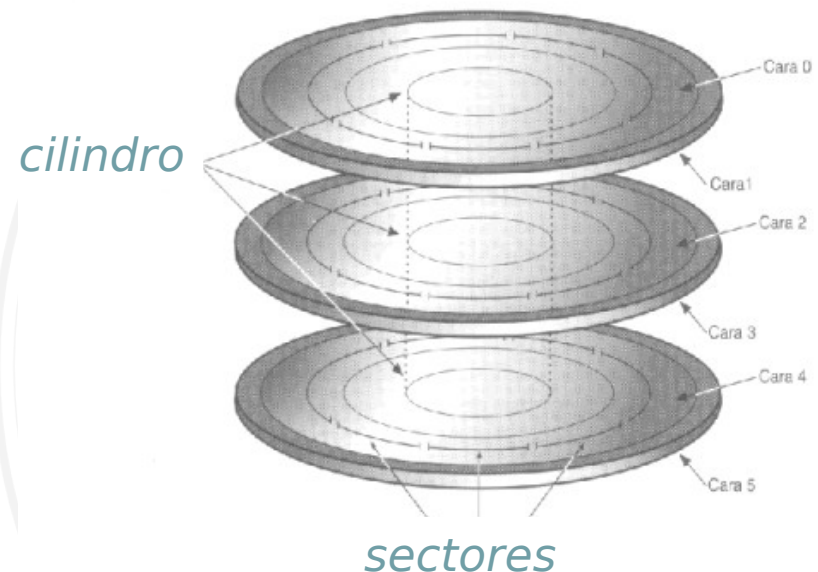
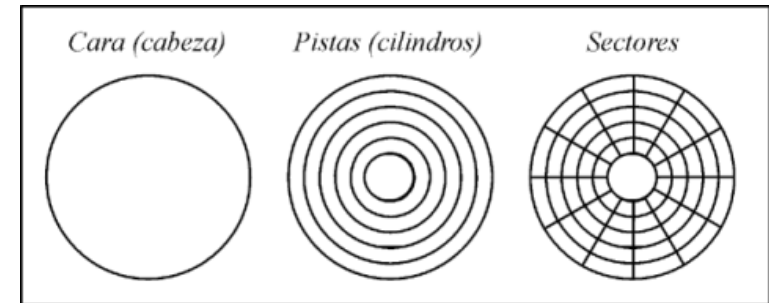
- Hace trabajos de **caché** de la **porción** de la **BD** de **uso más reciente**.
- Elemento de almacenamiento intermedio que **ubica** de forma **temporal** los **datos afectados por las operaciones**.

Como es rápida y cara (preciada), el nivel interno debe optimizar su uso para acelerar el procesamiento de los datos.

Tanto el disco duro como la memoria principal utilizan distintos niveles de caché para acelerar el acceso a los datos.

Discos Duros

- Dispositivo de almacenamiento **más usado** en **BD**.
- Constituidos por un **conjunto** de **discos magnéticos** con dos caras.
- **Cada** cara tiene un conjunto de **pistas** concéntricas (**cilindro**: la **misma pista** de **todas** las **caras**).
- Cada **pista** se divide en **sectores** con la misma capacidad de almacenamiento (**bloque**).
- **Localización** de un **bloque**:
 - **cilindro**
 - **superficie** de disco
 - **sector**
- **Parámetros**:
 - **Capacidad**
 - **Tiempo medio** de **acceso**
 - **RPM**
 - **Velocidad sostenida** de lectura/escritura.



Medidas de rendimiento:

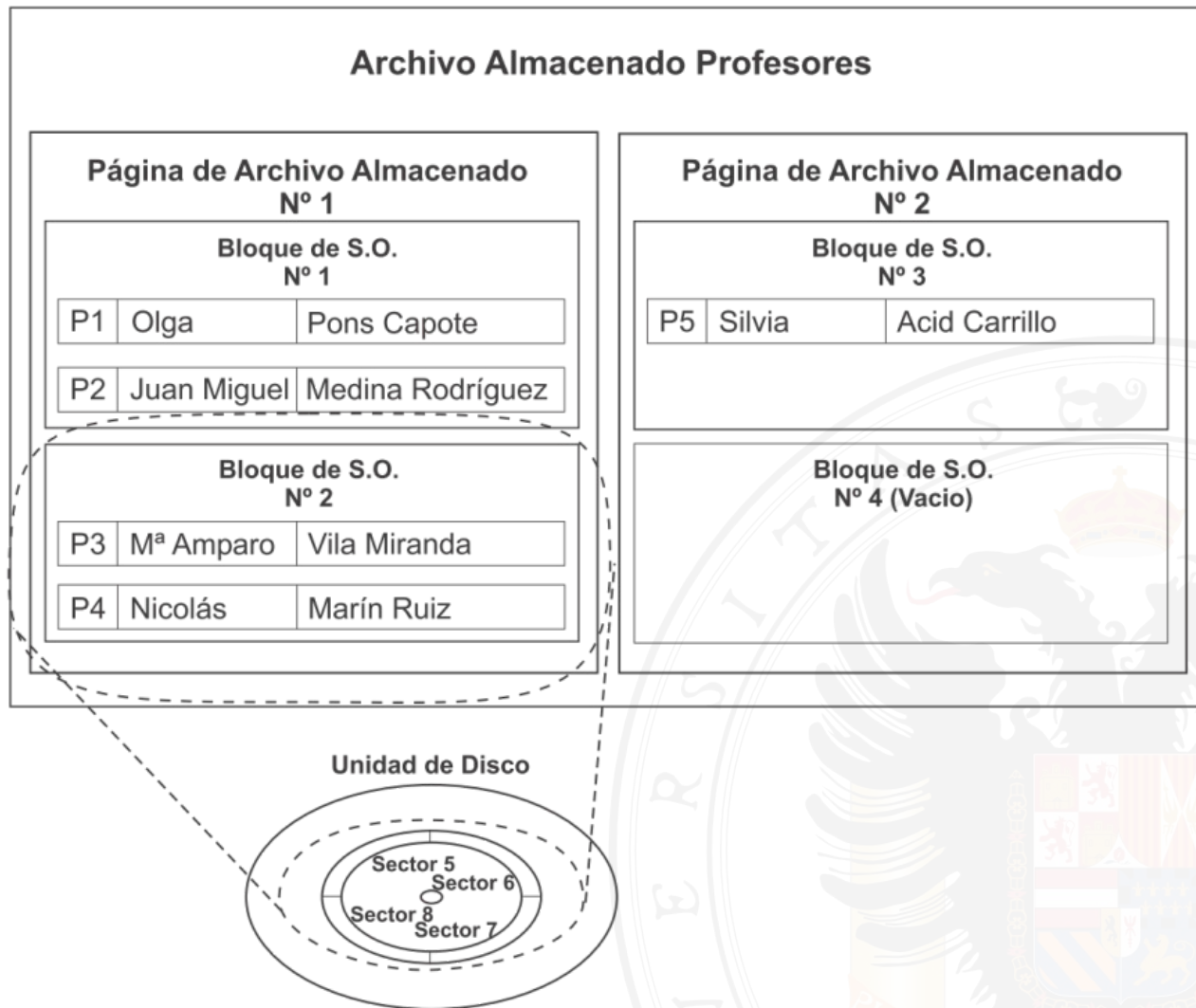
- Tiempo medio de acceso (t_a): tiempo medio transcurrido entre una instrucción y la obtención de la información
- Tiempo medio de búsqueda (t_b): tiempo medio de posicionamiento en pista
- Tiempo de latencia rotacional (t_l): tiempo medio de posicionamiento en sector

$$t_a = t_b + t_l$$

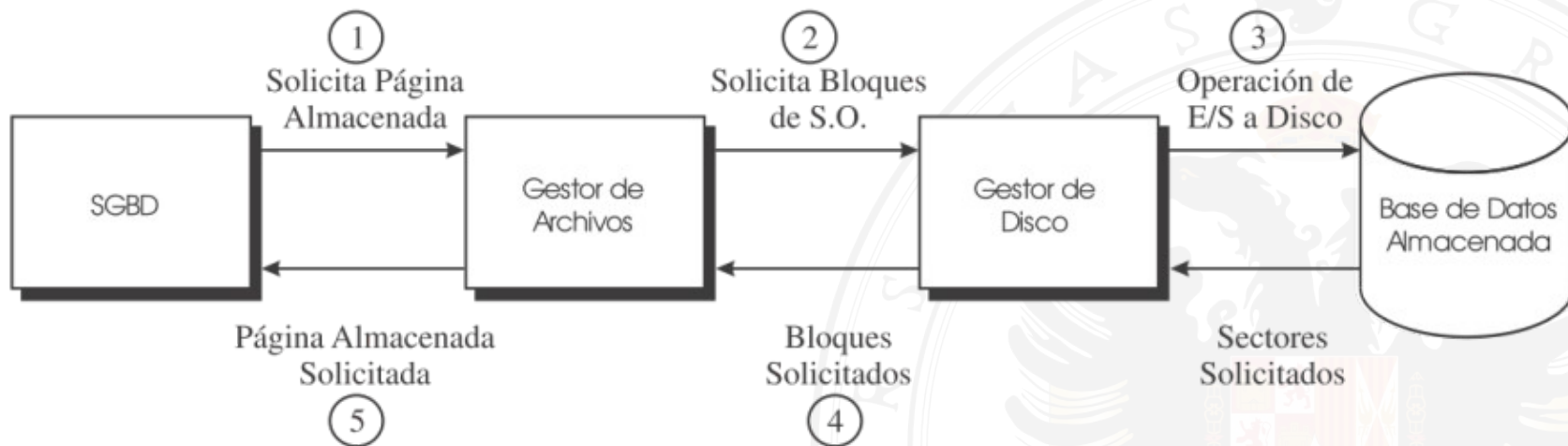
- Tiempo medio entre fallos (MTBF)

1. Conceptos básicos
2. Dispositivos de almacenamiento
3. **Métodos de acceso a la BD almacenada**
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD

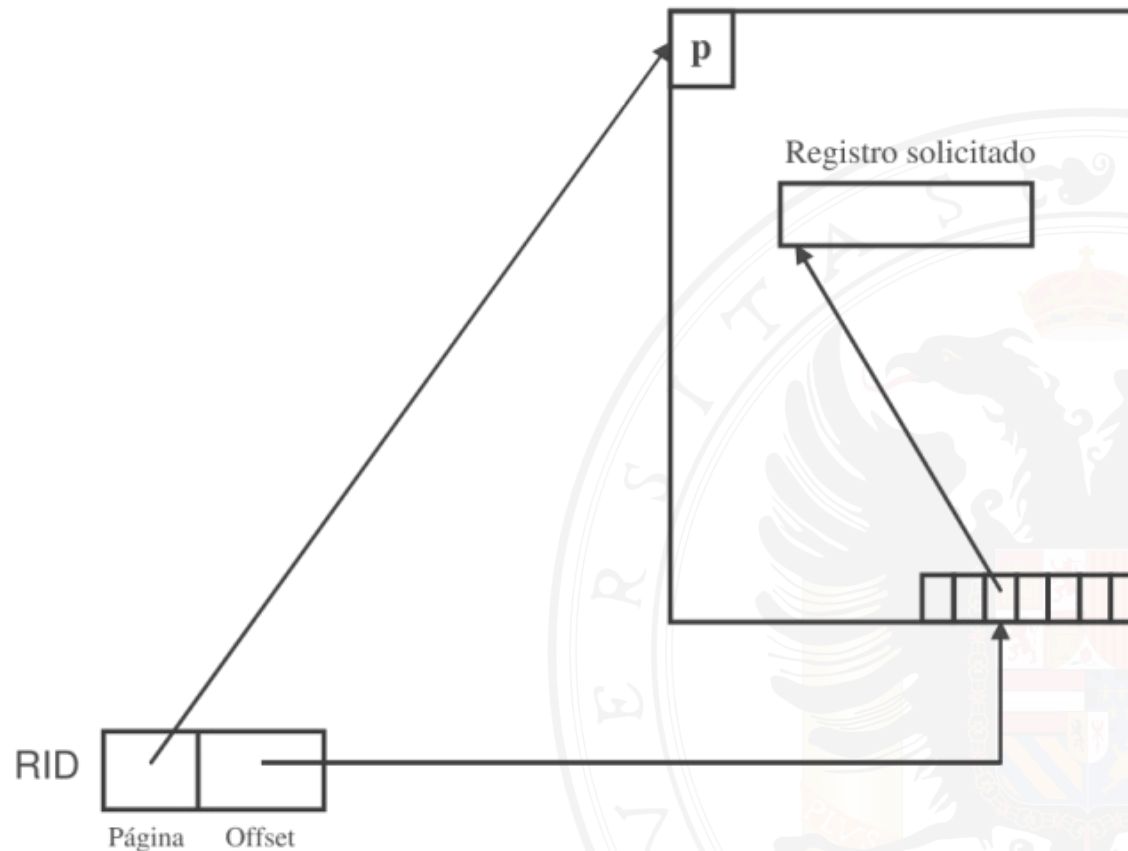




¿Cómo se transforma un registro almacenado en una representación física en el almacenamiento secundario?



Para que el gestor de almacenamiento pueda localizar un registro almacenado, se utiliza el RID (Record Identifier):



Cada registro almacenado tiene:

- Cabecera: Número y tipo de columnas que lo integran.
- Datos: Contenido de las columnas

Las páginas o bloques de la BD deben tener un tamaño múltiplo de las páginas del sistema operativo (mínima unidad de E/S).

Para recuperar un registro almacenado hay que determinar la página de BD que lo contiene y entonces recuperar los bloques de disco que la integran.

Hay que organizar la estructura de almacenamiento y los métodos de acceso, de forma que se optimice la interacción con los dispositivos de almacenamiento secundario.

Deben minimizarse las operaciones de E/S al almacenamiento secundario.

El gestor de disco del SO

Normalmente el SGBD interactúa con la BD almacenada en el sistema de almacenamiento secundario a través del **gestor de disco** del SO.

El gestor de disco organiza los datos en **conjuntos de bloques o archivos** de SO.

Una BD puede valerse de uno o varios de estos archivos para almacenar su contenido.

También se encarga de gestionar el **espacio libre en el disco**.

Funciones del **gestor de disco** del SO

- Crear un **nuevo archivo** de sistema operativo.
- Eliminar un **archivo** de sistema operativo existente.
- Añadir un **bloque** nuevo al **conjunto** de bloques **c**.
- Eliminar el bloque **b** del conjunto **de bloques c**.
- Devolver el bloque **b** del **conjunto de bloques c**.
- Reemplazar el bloque **b** **dentro** del conjunto de bloques **c**.

El gestor de archivos del SGBD

Componente del SGBD que se encarga de:

- Hacer la transformación entre:
 - Campos, registros y archivos almacenados a Bloques y conjuntos de bloques que pueda entender el gestor de disco.
- Organizar los datos de manera que se minimice el tiempo de recuperación. Minimizar las E/S a disco.

Funciones del gestor de archivos del SGBD (I)

- Crear un nuevo archivo almacenado.
 - Asociar al archivo un conjunto de páginas o bloques de la BD.
- Eliminar un archivo almacenado.
- Recuperar el registro almacenado r del archivo almacenado a .
 - Normalmente, el SGBD proporciona el RID.
 - Sólo hay que obtener en memoria la página que contiene el registro para extraerlo.
- Añadir un nuevo registro almacenado al archivo almacenado a . Hay que localizar la página de BD más apropiada de las pertenecientes al archivo almacenado.
 - Si no se pudiera, se solicita una nueva página.
 - Se devuelve al SGBD el RID nuevo.

Funciones del gestor de archivos del SGBD (II)

- Eliminar el registro r del archivo almacenado a .

Hay que recuperar la página de BD que contiene dicho registro y marcar el espacio ocupado por el registro en dicha página como disponible.

- Actualizar el registro r en el archivo almacenado a .

Recupera la página de la BD que contiene el registro que se desea actualizar.

Trata de sustituir la información. Si no puede, se intenta ubicar en otra página

1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



La BD se representa de diferentes formas en los diferentes niveles de la arquitectura del SGBD.

- Su representación en el **nivel interno** no tiene por qué **coincidir** con su representación en **el nivel conceptual**.
- Cada **conjunto de registros** del mismo tipo **no tiene por qué ser un mismo archivo**.

El nivel interno debe traducir las estructuras del nivel conceptual a otras estructuras intermedias más cercanas al almacenamiento real de los datos (nivel físico).

Agrupamiento

La BD en el nivel interno

Conjunto de páginas en las que se van ubicando los registros.

Agrupamiento Intra-Archivo

Ubicar en una página registros del mismo tipo.

Es el más frecuente.

Agrupamiento Inter-Archivo

Ubicar en una página registros de distinto tipo.

Ha de existir relación (por ejemplo entidades fuerte-débil)

Ejemplo

Se inserta una nueva asignatura con código A6.

- Se **localiza** la **primera página libre** (la 24).
- Se **inserta** el **registro** correspondiente.
- Se **añade esta página** al conjunto de **páginas de asignaturas**.

Se borra la asignatura con código A6.

- La **página** que contiene a esta asignatura **pasa** al conjunto de **páginas libres**.

Se introduce un nuevo profesor con código P7.

- Se **ubica** en la **primera página libre** disponible (la segunda).

Se borra A4.

- Su página pasa **al conjunto de páginas libres**.

Ejemplo

Se inserta una nueva asignatura con código **A6**.

- Se localiza la primera página libre (la **24**).
- Se inserta el registro correspondiente.
- Se añade esta página al conjunto de páginas de asignaturas.

0	1	2	3	4	5	24
	A₁	A₂	A₃	A₄	A₅	
6	7	8	9	10	11	
P₁	P₂	P₃	P₄	P₅	P₆	
12	13	14	15	16	17	
A₁/P₁	A₁/P₂	A₁/P₃	A₁/P₄	A₁/P₅	A₁/P₆	
18	19	20	21	22	23	
A₃/P₁	A₃/P₂	A₃/P₆	A₄/P₂	A₄/P₄	A₄/P₅	
24	X	25	26	27	28	29
A6						

Ejemplo

0	X	1	2	2	3	3	4	4	5	5	X
		A₁		A₂		A₃		A₄		A₅	
6	7	7	8	8	9	9	10	10	11	11	X
P₁		P₂		P₃		P₄		P₅		P₆	
12	13	13	14	14	15	15	16	16	17	17	18
A₁/P₁		A₁/P₂		A₁/P₃		A₁/P₄		A₁/P₅		A₁/P₆	
18	19	19	20	20	21	21	22	22	23	23	X
A₃/P₁		A₃/P₂		A₃/P₆		A₄/P₂		A₄/P₄		A₄/P₅	
24	25	25	26	26	27	27	28	28	29	29	X

Ejemplo

0	X	1	2	2	3	3	4	4	5	5	24
		A₁		A₂		A₃		A₄		A₅	
6	12	7	8	8	9	9	10	10	11	11	X
		P₂		P₃		P₄		P₅		P₆	
12	18	13	14	14	15	15	16	16	17	17	18
		A₁/P₂		A₁/P₃		A₁/P₄		A₁/P₅		A₁/P₆	
18	25	19	20	20	21	21	22	22	23	23	X
		A₃/P₂		A₃/P₆		A₄/P₂		A₄/P₄		A₄/P₅	
24	X	25	26	26	27	27	28	28	29	29	X
A₆											

Ejemplo

Se borra la asignatura con código **A2**.

- La página que contiene a esta asignatura pasa al conjunto de páginas libres.
- Se reorganiza la lista correspondiente a Asignaturas.

0	1	3	2	25	3	4	5	24
	A₁				A₃	A₄	A₅	
6	7	8	9	10	11			
P₁	P₂	P₃	P₄	P₅	P₆			
12	13	14	15	16	17			
A₁/P₁	A₁/P₂	A₁/P₃	A₁/P₄	A₁/P₅	A₁/P₆			
18	19	20	21	22	23			
A₃/P₁	A₃/P₂	A₃/P₆	A₄/P₂	A₄/P₄	A₄/P₅			
24	X	25	26	27	28			X
A₆								

Ejemplo

Se introduce un nuevo profesor con código **P7**.

- Se ubica en la primera página libre disponible (la segunda).
- Se ajustan las cadenas de punteros.

0	1	3	2	2	3	4	5	24
	A₁	P7		A₃	A₄	A₅		
6	7	8	9	10	11	2		
P₁	P₂	P₃	P₄	P₅	P₆			
12	13	14	15	16	17			
A₁/P₁	A₁/P₂	A₁/P₃	A₁/P₄	A₁/P₅	A₁/P₆			
18	19	20	21	22	23			
A₃/P₁	A₃/P₂	A₃/P₆	A₄/P₂	A₄/P₄	A₄/P₅			
24	X	25	26	27	28	29	X	
A₆								

Ejemplo



Se borra **A4**.

- Su página pasa al conjunto de páginas libres.
- Se reorganiza la cadena de punteros de las Asignaturas.

0	1	3	2	X	3	5	4	25	5	24
	A₁	P₇	A₃						A₅	
6	7	8	9	10	11	2				
P₁	P₂	P₃	P₄	P₅	P₆					
12	13	14	15	16	17					
A₁/P₁	A₁/P₂	A₁/P₃	A₁/P₄	A₁/P₅	A₁/P₆					
18	19	20	21	22	23					
A₃/P₁	A₃/P₂	A₃/P₆	A₄/P₂	A₄/P₄	A₄/P₅					
24	X	25	26	27	28	29	X			
A₆										

Ejemplo

Punteros para el recorrido secuencial lógico

0	X	1	3	2	X	3	5	4	25	5	24
A₁			P₇			A₃			A₅		
6	7	7	8	8	9	9	10	10	11	11	2
P₁		P₂		P₃		P₄		P₅		P₆	
12	13	13	14	14	15	15	16	16	17	17	18
A₁/P₁		A₁/P₂		A₁/P₃		A₁/P₄		A₁/P₅		A₁/P₆	
18	19	19	20	20	21	21	22	22	23	23	X
A₃/P₁		A₃/P₂		A₃/P₆		A₄/P₂		A₄/P₄		A₄/P₅	
24	X	25	26	26	27	27	28	28	29	29	30
A₆											

Ejemplo Índice de la página 0

0		X
---	--	---

Conjunto	Dirección a Primera Pág
Pág. Libres	4
Asignaturas	1
Profesores	6
Imparte	12

Ejemplo Factor de bloqueo N

p	Información de cabecera		
A1	BASES DE DATOS	A2	ALGEBRA
A3	COMPUTABILIDAD	A4	METODOLOGÍA
A5	PROGRAMACION DE BD	Espacio libre	

En realidad, las **páginas** contienen **más de un registro**

Ejemplo

p	Información de cabecera		
A1	BASES DE DATOS	A3	COMPUTABILIDAD
A4	METODOLOGÍA	A5	PROGRAMACION DE BD
A7	MATEMÁTICA DISCRETA	A9	SISTEMAS DE BD
Espacio libre			

Insertión de A9, borrado de A2 e inserción de A7

Consideraciones finales

La organización descrita es un **ejemplo general**. Cada SGBD comercial utiliza su variante concreta, aunque la idea subyacente es la misma.

No existe una relación **directa fichero-almacenado/fichero-físico**, ya que todos los conjuntos de páginas irán almacenados, con toda probabilidad, en uno o varios ficheros físicos.

1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



OBJETIVO: Minimizar el número de accesos a disco → **minimizar la cantidad de páginas de BD involucradas en una operación de BD.**

- Ninguna de las organizaciones presentadas es **mejor** en **términos absolutos**.
- Criterios básicos para medir la “**calidad**” de una organización son:
 - **Tiempo de acceso** a los datos requeridos.
 - Porcentaje de **memoria** ocupada por los **datos requeridos** con **respecto** a las **páginas de BD** que los contienen.
- Trabajaremos a dos niveles:
 - Organización de **registros** de datos a **nivel de almacenamiento**.
 - Adición de **estructuras** complementarias para **acelerar** el **acceso** a dichos registros.

1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



Fichero de acceso secuencial:

- Aquél donde los **registros** están almacenados **consecutivamente**.
- Para acceder a un **registro determinado** debemos **pasar** obligatoriamente por los **registros que le preceden**.
- Los registros suelen estar **ordenados** por una clave (**clave física**).

Número de bloque	clave de búsqueda ↙	Número de registro relativo
0	07	0
	10	1
	13	2
		3
1		4
	20	5
	23	6
	25	7
	26	

Ejemplo:

- Mostrar la **relación completa** de departamentos.
- La consulta se resolvería **rápidamente** si los departamentos están **almacenados conjuntamente** en **bloques contiguos** de un fichero.
- Sin embargo:
 - ¿Qué pasa si queremos plantear **consultas por valor de clave** o por **rango de valores**?

El primer caso implica:

- Recorrer **uno tras otro** cada uno de **los registros**.
- En el **caso peor** (no encontrarse dicho departamento o ser el último de la lista) supone **recorrer de forma completa el fichero**.
- Búsqueda es $O(N)$.

El segundo caso tiene un tratamiento muy parecido:

- Se realiza la **búsqueda** por valor de clave de la **cota inferior** del intervalo.
- Se continúa **hasta alcanzar la cota superior**. Si están ordenados por el valor de la clave.

Inserción de un nuevo registro:

- Buscar el **bloque** que le corresponde.
 - Si hay **sitio**, **se inserta** el nuevo registro.
 - En caso **contrario**, o bien se opta por crear un **nuevo bloque** o bien se crea un **bloque de desbordamiento**.
- Es **recomendable** dejar **espacio vacío en los bloques** para evitar los problemas de reorganización.

Borrado de un registro:

- Buscar el **registro**.
- Puede implicar una **reorganización local** de los registros de un bloque.

En resumen, las dos operaciones suponen:

- **Escritura** del **bloque** del registro que se inserta o borra.
- **Creación o liberación** de **bloques** de datos en el fichero secuencial.
- **Creación o liberación** de **bloques** de desbordamiento.
- **Reorganización** de **registros** entre **bloques contiguos**, lo que implica la escritura de los bloques implicados en el desplazamiento.

Como puede verse, esta forma de organizar los registros no está exenta de grandes inconvenientes.

Pueden subsanarse, al menos en parte, mediante el uso de **estructuras adicionales** que nos permitan:

- Acelerar la **localización** de los **datos**.
- Disminuir el **número de bloques de disco** transferidos.

Entre las técnicas más populares se encuentran:

- Índices.
- Acceso directo.

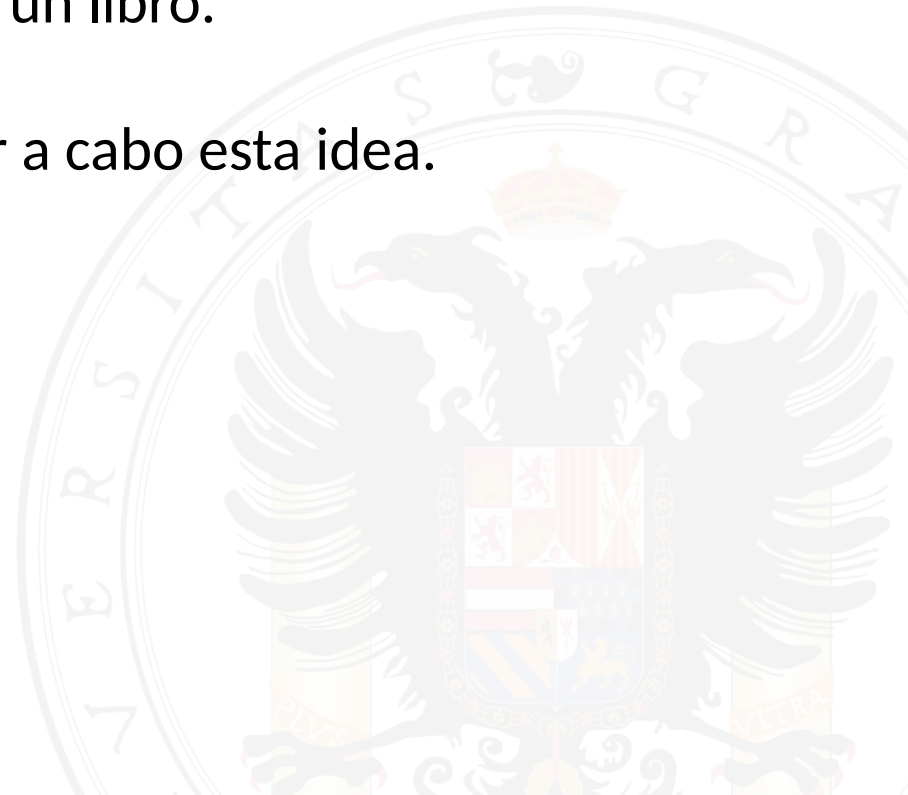
1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



Tiene por objeto **disminuir** el **tiempo de acceso a los datos** por una **clave** de búsqueda.

Similar a la idea de un índice en un libro.

Existen muchas formas de llevar a cabo esta idea.

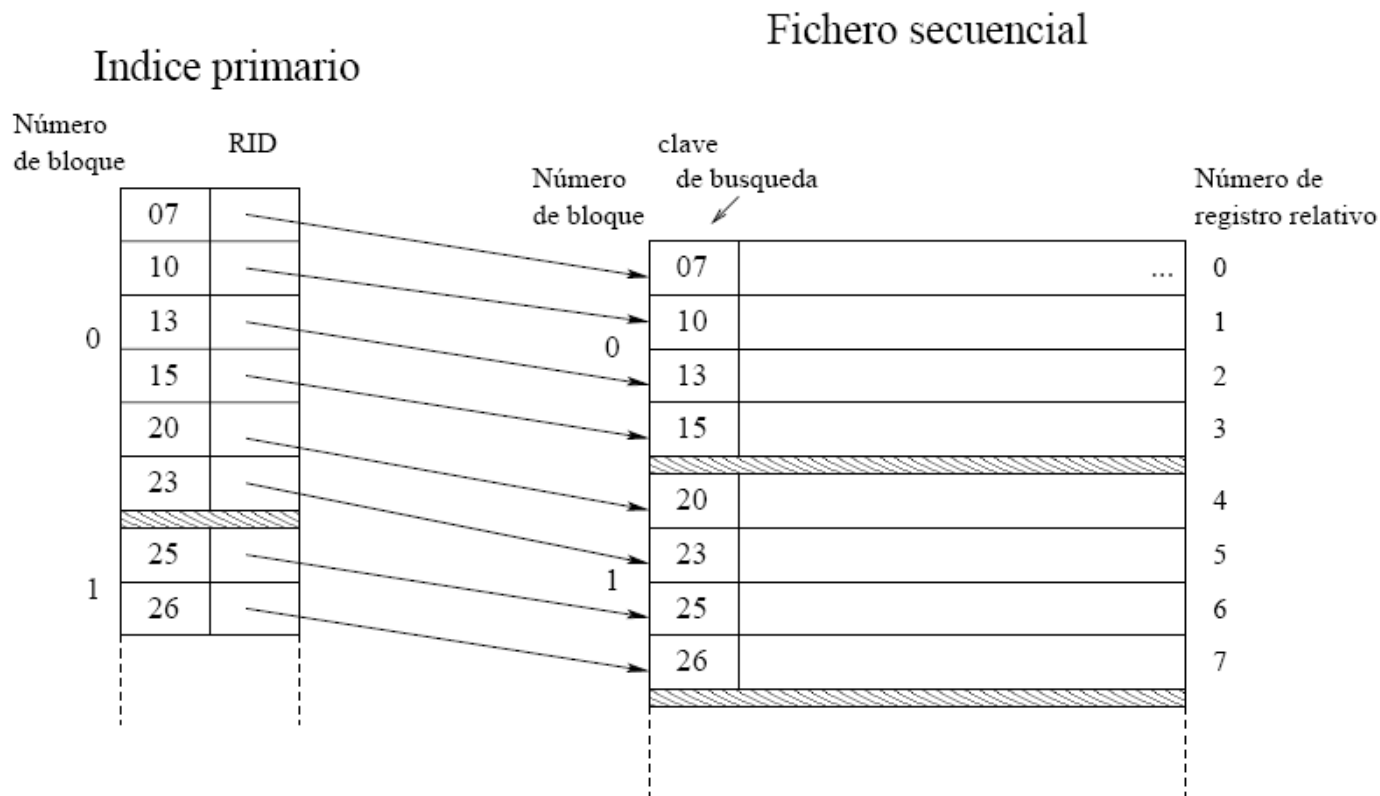


Ficheros indexados:

Partimos de un fichero secuencial.
Disponemos una estructura adicional: **fichero índice**.

- Sus **registros** poseen:
 - **Campo clave** (la clave de búsqueda).
 - **Campo** de referencia que contiene **RIDs** de registros.
- Son **más pequeños** que los del fichero de datos, aunque el **número de ellos es el mismo** en ambos ficheros.

Ficheros indexados:



Ficheros indexados:

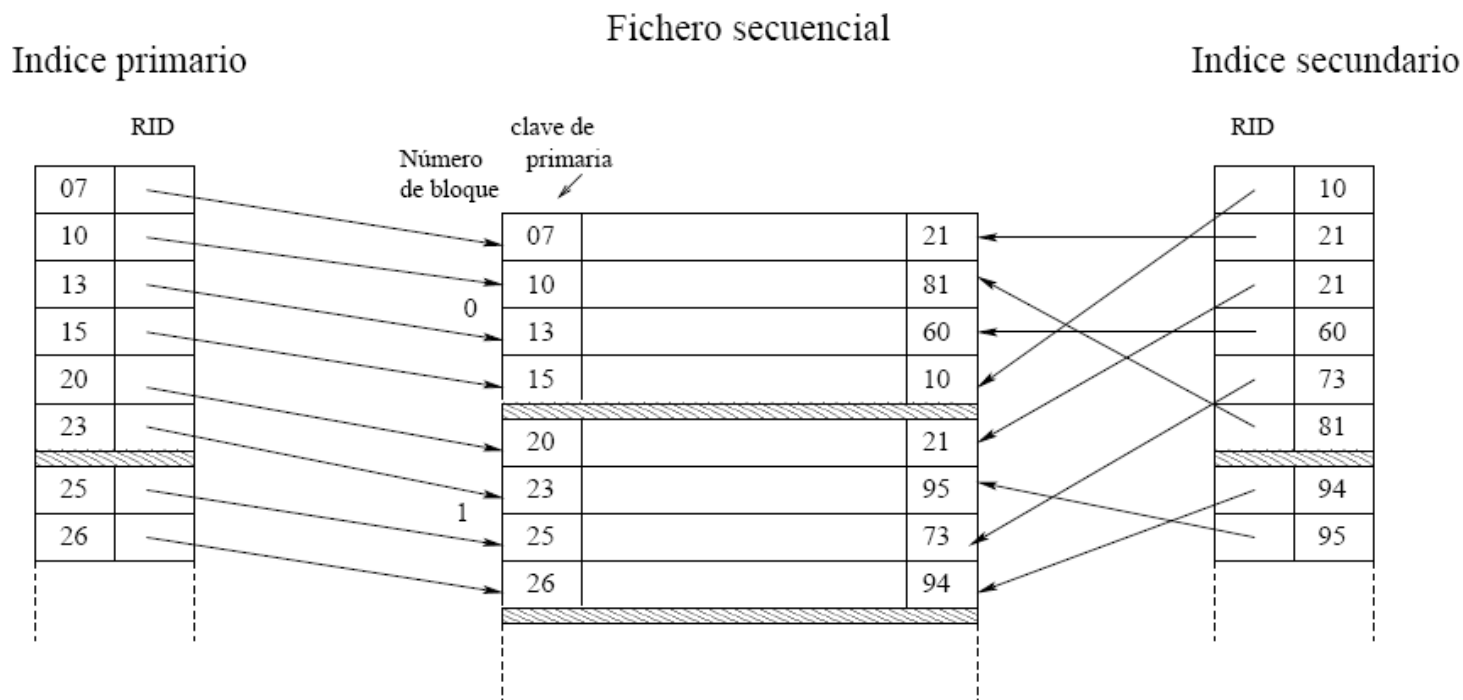
Índice primario:

- La **clave** de búsqueda es el **mismo campo** clave por el que está ordenado el fichero de datos.

Índices Secundarios:

- Construidos sobre **otros campos** que **no** sean la **clave física** del fichero de datos.

Ficheros indexados:



Ficheros indexados:

Proceso de consulta:

- Consulta por un valor de la clave
 - Sobre el índice localizamos la clave (recorrido secuencial).
 - Obtenemos el RID del registro requerido.
 - Vamos a disco para recuperar el bloque de datos donde se encuentra el registro señalado por el RID.
 - La búsqueda en el índice es más rápida.
- Consulta por rango de valores:
 - Búsqueda en el índice por valor de clave de la cota inferior.
 - Recorrido de las entradas del índice que están en el intervalo, recuperando los registros correspondientes gracias a su RID.

Ficheros indexados:

Inserción de un nuevo registro:

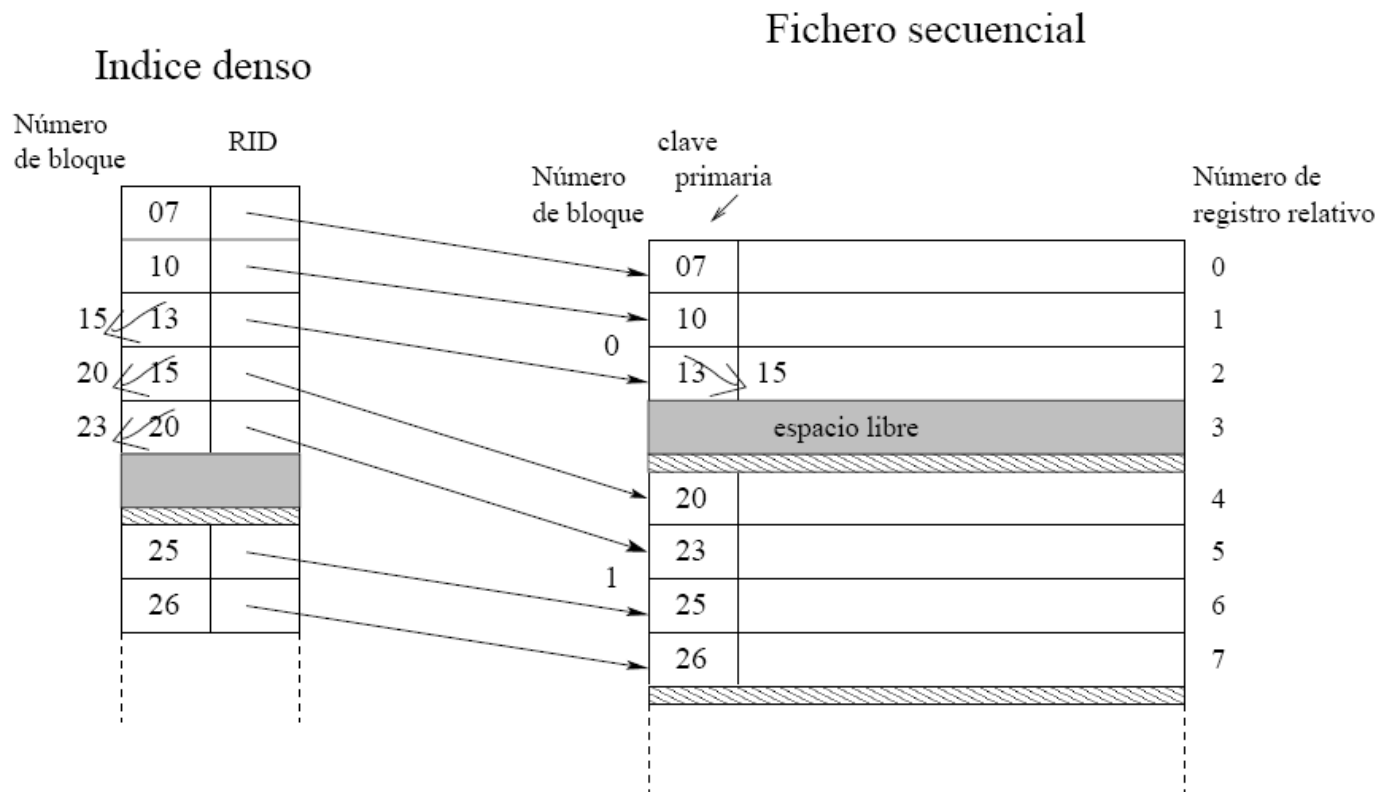
- Las mismas operaciones que en el fichero secuencial.
- Hay que actualizar también el índice (inserción en un fichero secuencial).

Borrado de un registro:

- Borrado de un registro en el fichero de datos.
- Borrado de una entrada en el índice.

Ficheros indexados:

Ejemplo: Borrado del registro con CP 13



Ficheros indexados:

Se puede montar un índice sobre más de un campo de un registro.

- **Clave:** concatenación de los campos indicados.

Hay que tener cuidado. Por ejemplo, un índice sobre *nombre-alumno* y *DNI*.

- **Útil** para consultas que involucran:
 - Nombre.
 - Nombre y DNI.
- **No es útil** para consultas sobre el DNI.

Ficheros indexados:

Conclusiones:

- Los **índices**:
 - **Aceleran** el **acceso** a los datos.
 - **Ralentizan** las **otras operaciones**.
 - Hay que **mantener** el **índice**.
- Por tanto:
 - Hay que **considerar** la **conveniencia** de crear cada **índice**.
 - **Frecuencia** de las **consultas**.
 - **Frecuencia** de las **operaciones** de **mantenimiento** de los datos.

1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



Lo Ideal: Mantener el índice en memoria principal.

La Realidad: los índices siguen siendo muy grandes, porque contienen todos los registros del fichero que indexan.

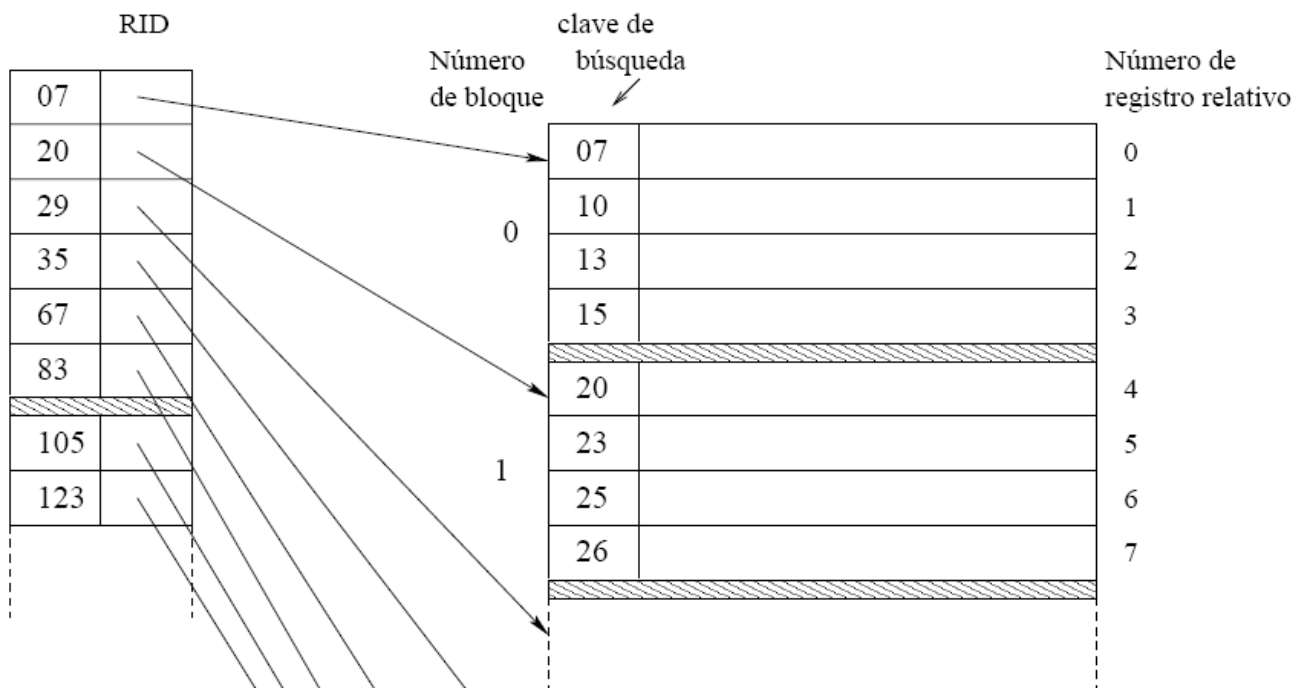
- Son densos.

Para reducir el tamaño aparecen los índices no densos:

- Registros compuestos por:
 - La clave de búsqueda.
 - La dirección de comienzo del bloque donde puede encontrarse el registro deseado.
- El número de registros se reduce al número de bloques del fichero de datos.
 - El acceso secuencial al índice no denso se acelera.

Índice no denso

Fichero secuencial



Diferencias en el proceso de búsqueda:

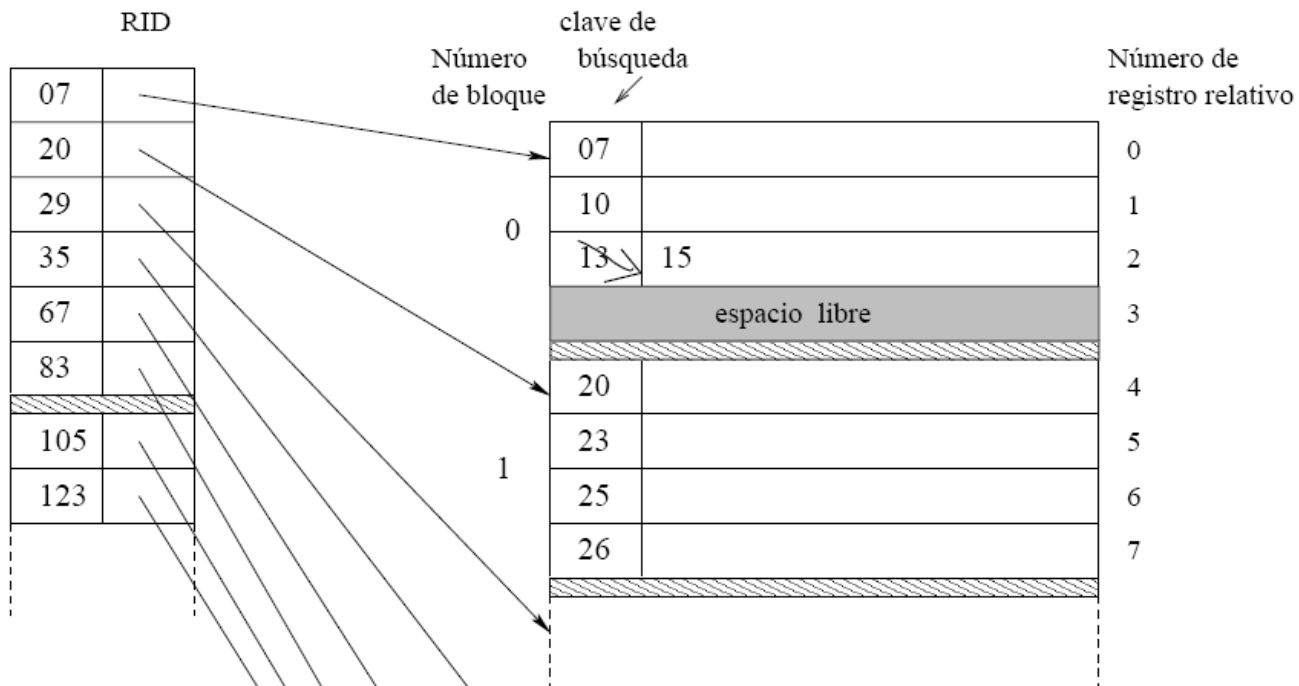
- Una vez **encontrado** el **bloque** donde **podría** encontrarse el registro:
 - Hay que **cargarlo en memoria**.
 - Hay que hacer una **búsqueda secuencial**.
 - No tiene **costes** en términos de **acceso a disco**.
- No se tiene **garantía** alguna de **encontrar** el **registro** deseado hasta consultar el **bloque** de datos **leído**.

Los índices no densos sólo se pueden definir sobre la clave física.

- El mantenimiento de un índice no denso es menos costoso:
 - Inserción y borrado menos frecuentes
 - Sólo ocurren cuando la operación afecta al valor representativo del bloque.

Índice no denso

Fichero secuencial



1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



Volvemos al objetivo de disminuir el tiempo necesario para recorrer el índice en busca de un registro:

- Idea: crear **índices sobre índices**.
- **Varios niveles** en el acceso a los datos.

Un índice multinivel está formado por:

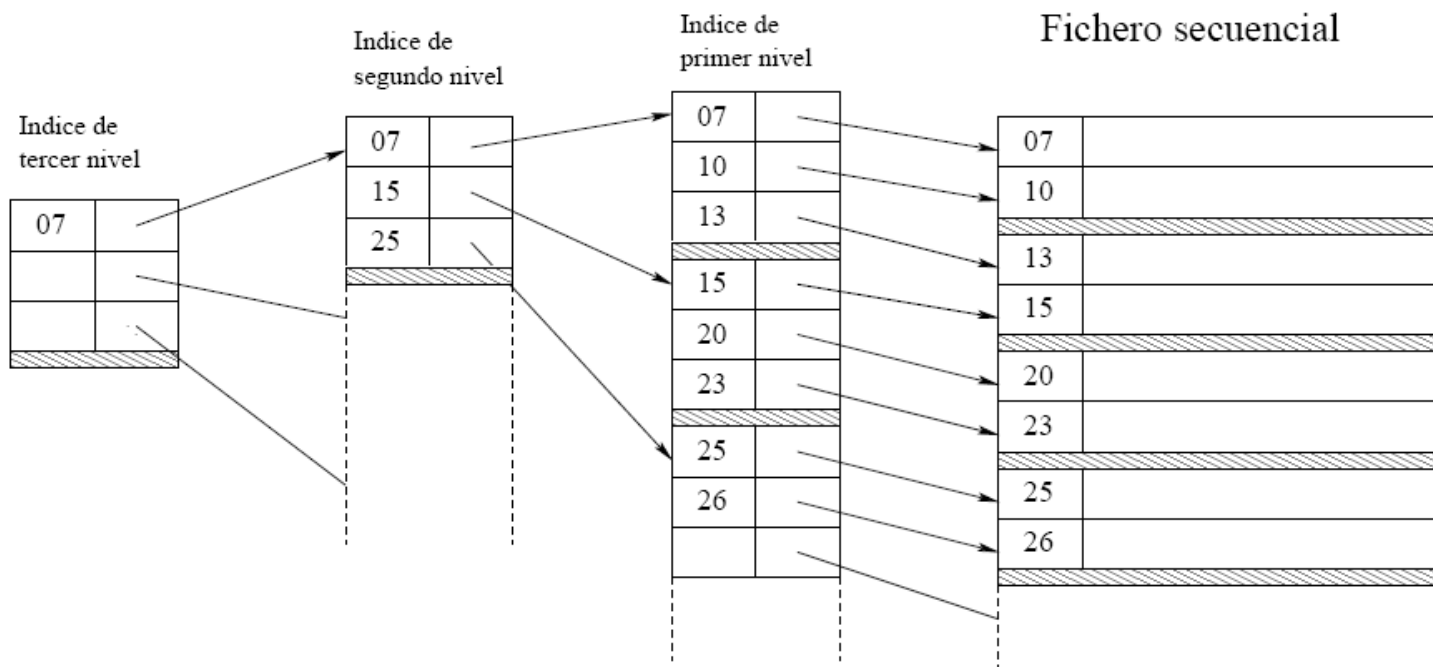
- Un **índice** de primer nivel **sobre el fichero de datos**.
 - Puede ser **denso o no dependiendo** de la **clave**.
- **Otros índices, no densos**, contruidos sucesivamente **unos sobre otros**.

El **tamaño de los bloques** se establece con la idea de optimizar cada una de las **operaciones de acceso al disco físico**.

Se **reduce el número de accesos a disco** para localizar un registro:

- En el peor caso: **tantos como niveles**

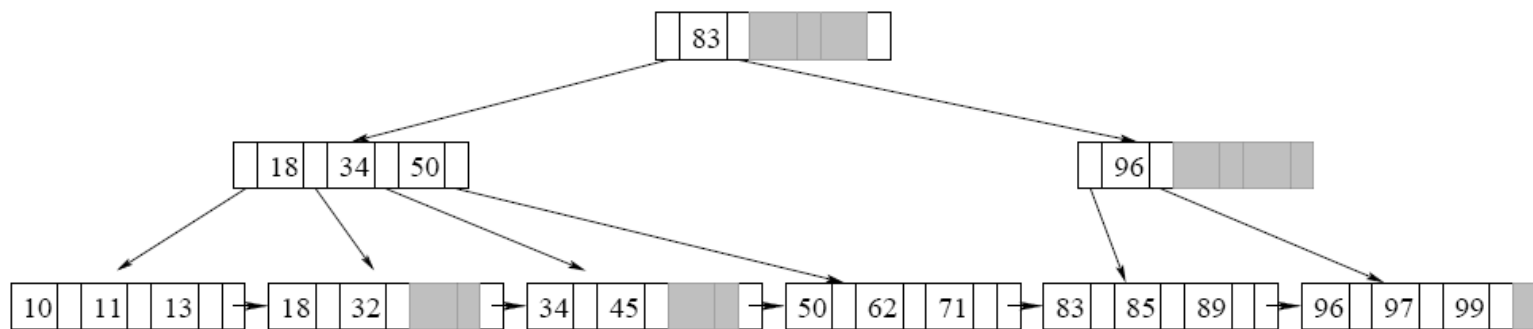
Se **complica el mantenimiento del índice**.



1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD

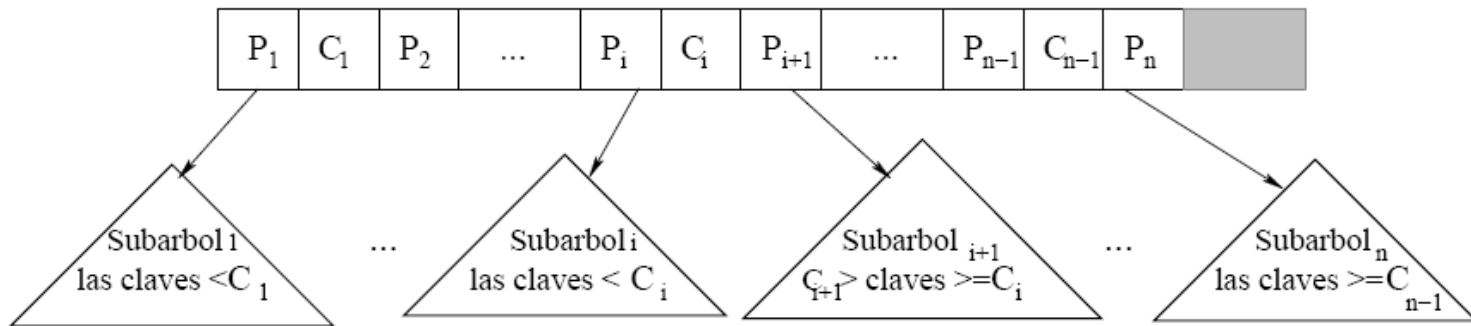


- Propuestos en 1972 por Bayer y McCreight son una **generalización** de los **árboles binarios balanceados** en la que los nodos pueden tener **más de dos hijos**.
- Todos los **valores de la clave** se encuentran almacenados en los **nodos hoja**.
- Un Árbol B+ de **orden M** (el **máximo** número de **hijos** que puede tener **cada nodo**) es un árbol con la siguiente estructura:
 - Nodo de nivel superior: **raíz del árbol**.
 - Nodos del nivel inferior: **hojas**.
 - Cada **nodo distinto de las hojas** tiene como **máximo M hijos**.
 - Cada **nodo** (excepto raíz y hojas) tiene como **mínimo $(M+1)/2$ hijos**.
 - La **raíz tiene al menos 2 hijos si no** es un nodo **hoja**.
 - Todos los **nodos hoja** aparecen al **mismo nivel**.
 - Las **claves** contenidas en **cada nodo** nos **guiarán** hasta el **siguiente nodo** del **nivel inmediatamente inferior**.
 - Un **nodo no hoja con n hijos** contiene:
 - **n-1 valores de clave** almacenados.
 - **n punteros P_i** que **apuntan** a un **nodo hijo**.



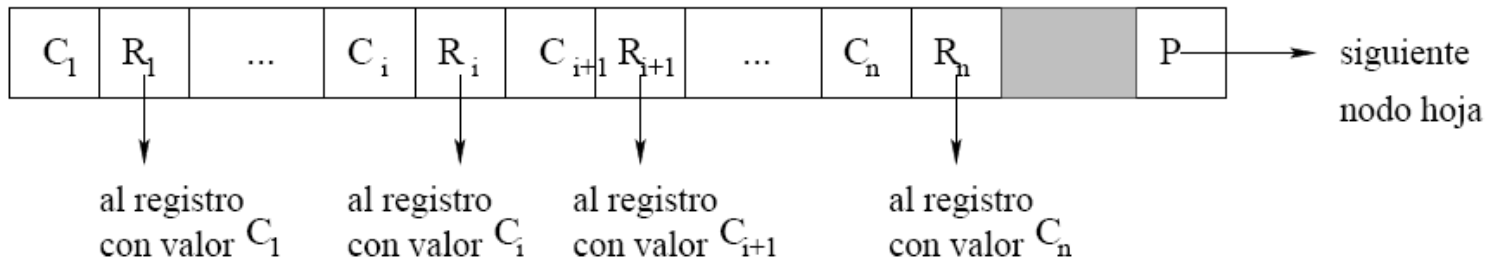
Restricciones dentro de los nodos:

- Los valores de clave C_i están ordenados dentro del nodo.
- Los valores x del subárbol apuntado por P_i cumplen:
 - $C_{i-1} \leq x < C_i$
 - Excepto para:
 - $i = 1$, donde $x < C_1$
 - $i = m$, donde $x \geq C_m$



Nodos hoja:

- Tienen una estructura diferente:
 - Parejas **clave – RID**.
 - **Punteros** al siguiente nodo hoja.
 - Algunas variantes también tienen punteros al nodo hoja anterior.
- La lista concatenada de nodos hoja (**conjunto secuencia**), tiene gran utilidad a la hora de hacer **consultas por intervalos**.



Restricciones en nodos hoja:

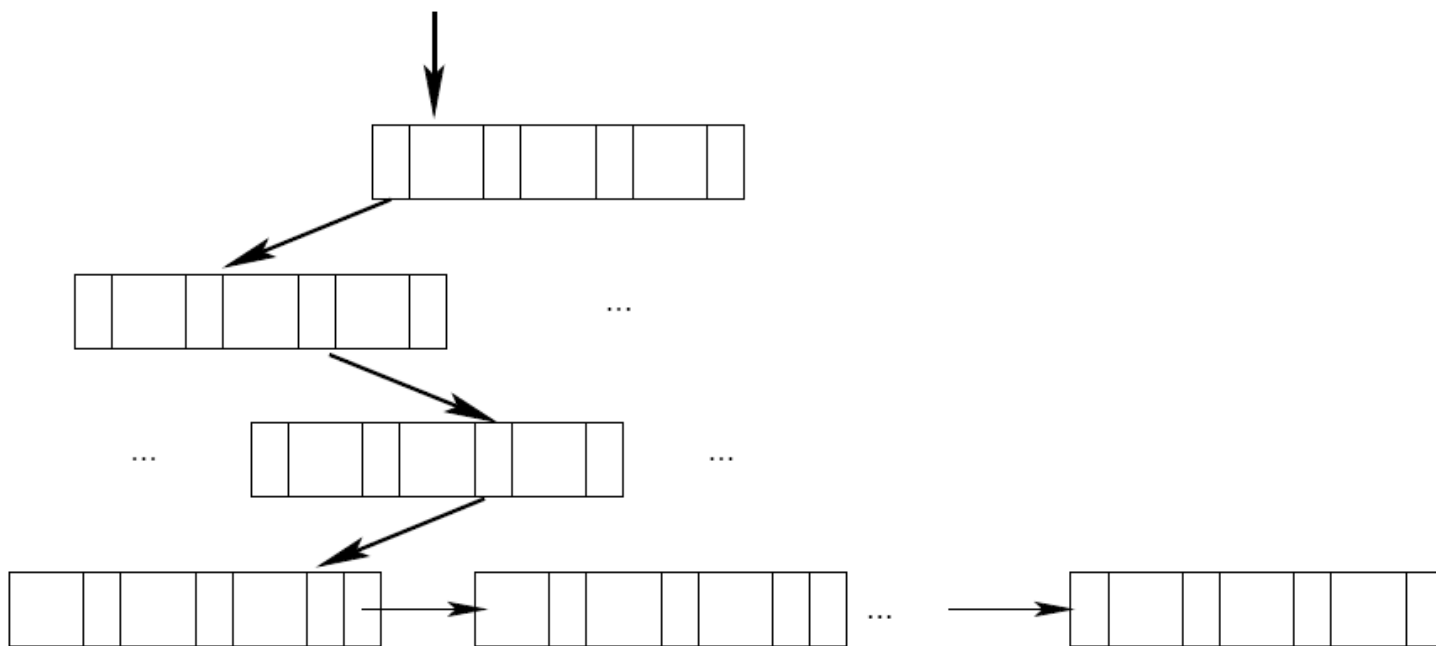
- Las **claves** aparecen **ordenadas** en cada **nodo**.
- Todas **las claves** han de ser **menores que** las del **siguiente nodo** en el conjunto secuencia.
- Los **nodos** han de estar como **mínimo rellenos hasta la mitad**.
- Todos los **nodos hoja** se encuentran en el **mismo nivel**.
 - Árbol **equilibrado**.
 - Todos los **caminos desde la raíz a un nodo hoja** tienen la **misma longitud**.

Proceso de consulta

- Localización de **un registro**:
 - Navegamos desde la **raíz**, bajando niveles.
 - Buscamos el registro en el **nodo hoja** y, en su caso, recuperamos el registro del fichero de datos gracias al **RID**.
- Consultas por **rango**:
 - Se localiza el **nodo hoja** que contiene el **valor inferior**.
 - Se recorren los **nodos hoja hasta** alcanzar el **superior**, recuperando los registros pertinentes del fichero de datos

Inserción y borrado

- Se utilizan algoritmos que garantizan que el árbol resultante sea equilibrado.



1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



- Los **Árboles-B** (B-Tree) son una **variante** de **B+Tree** en la que **no** almacenan todos los valores de la clave en los **nodos hoja**, si no que algunos valores se van almacenando en los **nodos intermedios** conforme se crea el árbol.
- Programa Java con animaciones interactivas para ilustrar el funcionamiento de los Árboles B en [Gnarley trees](https://people.ksp.sk/~kuko/gnarley-trees/) (Java requerido).

<https://people.ksp.sk/~kuko/gnarley-trees/>

- Ejecutar desde una consola:

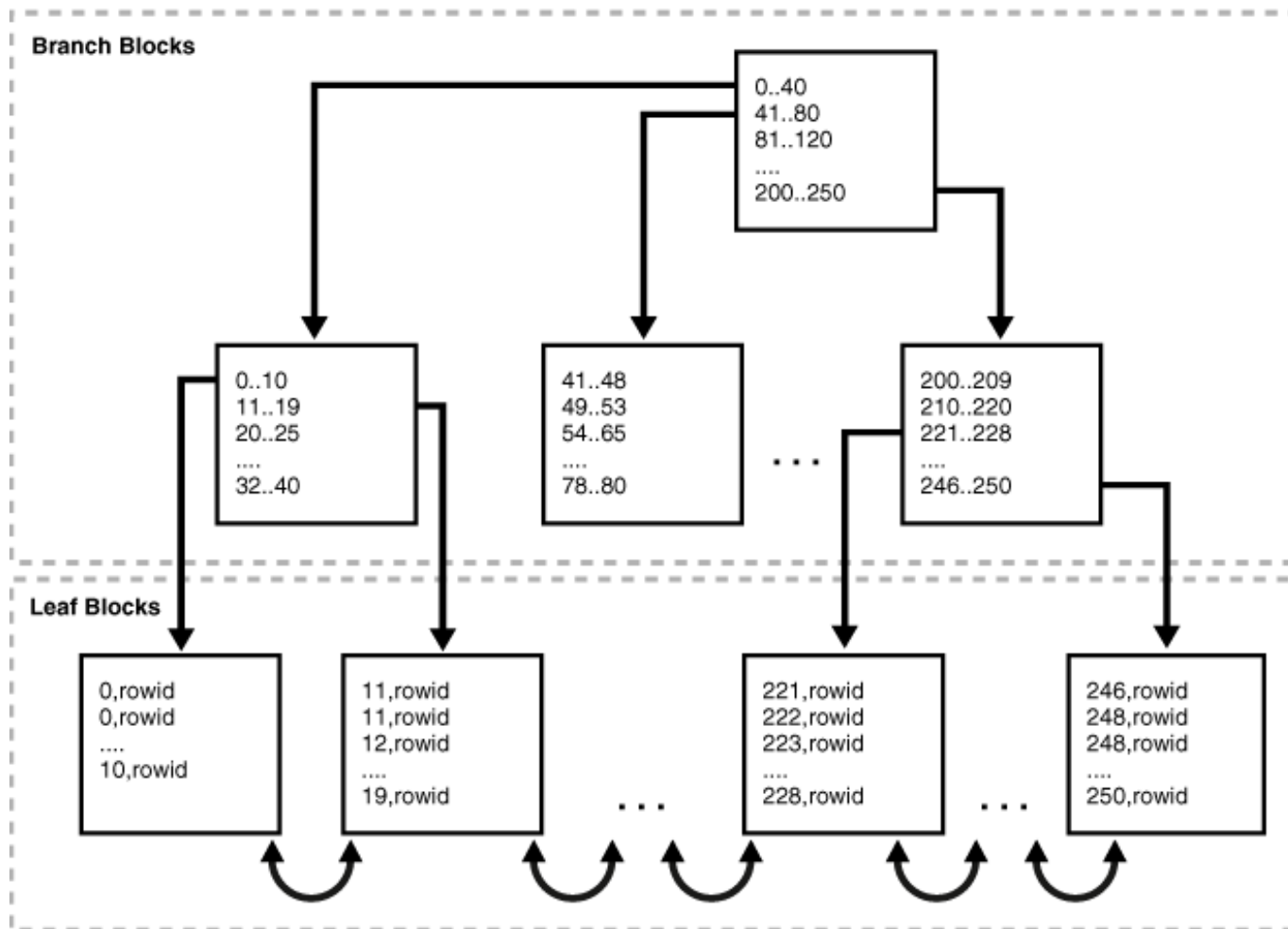
```
java -jar gt.jar
```

y seleccionad la pestaña B-tree en esta animación.

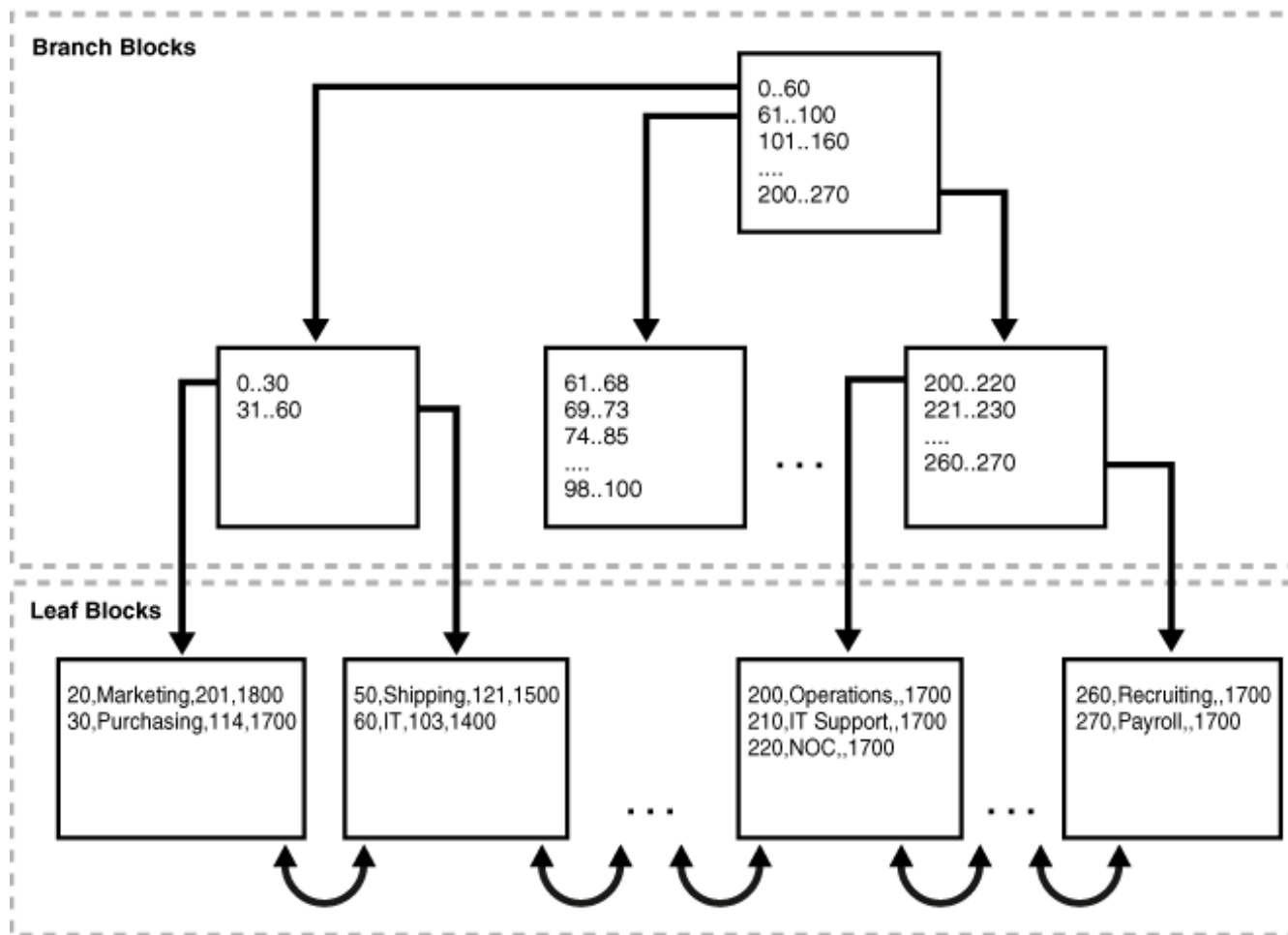
1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. **Uso de Árboles B+ en BD**
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



- Son **variaciones** del **Árbol B+**, de **orden elevado**, en la que se procura que **cada nodo** tenga una **capacidad** de almacenamiento **similar** al **tamaño** de un **bloque de datos**.
- Esto **reduce** los **accesos a disco** que suelen ser los que determinan el rendimiento de las búsquedas en BD.
- En los **nodos intermedios** sólo **están** los **rangos** de los valores de la **clave** y los **punteros** a los **nodos hijo correspondientes**.
- En los **nodos hoja** se encuentran **todos** los valores de la **clave** **ordenados junto** con los **RIDs(rowid)** que **apuntan** a las **tuplas** que contienen ese valor de la clave.
- Los **nodos hoja**, que forman el **conjunto secuencia**, se encuentran **enlazados** para poder recuperar por búsquedas secuenciales, a veces se encuentran **doblemente enlazados**, para **facilitar búsquedas** **ascendentes y descendentes** por el valor de la clave.



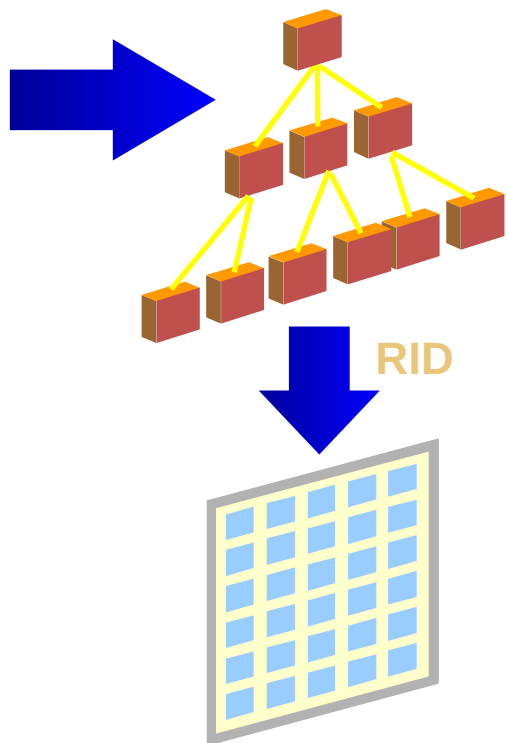
Tablas Organizadas por Índice (IOT). Las hojas contienen las tuplas en lugar del RID. Una IOT sólo puede estar organizada de esta forma mediante una clave, aunque se pueden definir índices adicionales basados en otras claves.



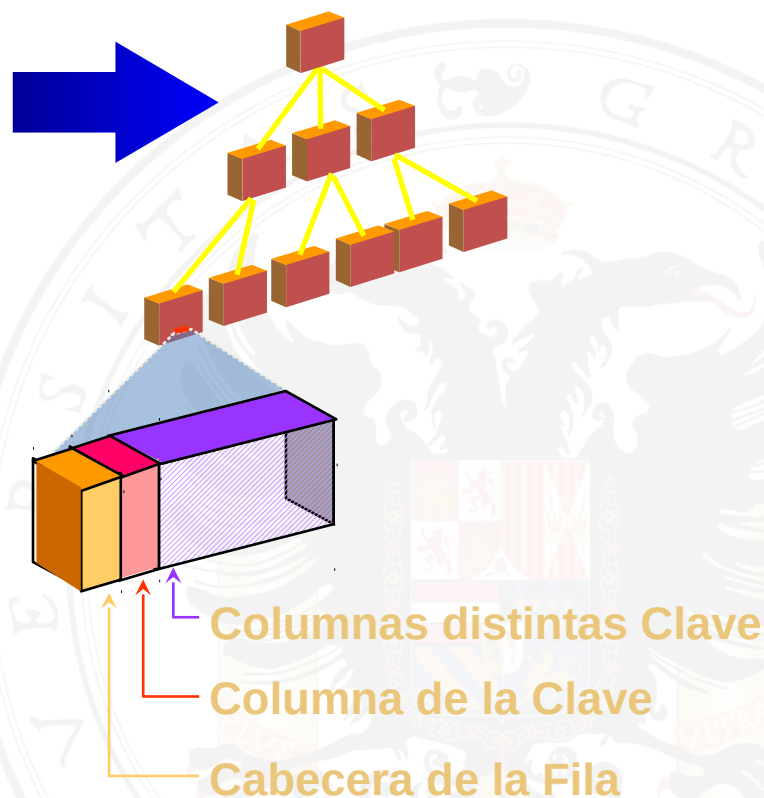
Tablas Organizadas por Índice (IOT). Estructura comparada con la de los índices normales.

```
CREATE TABLE proyecto(codpj CHAR(3) PRIMARY KEY, nompj VARCHAR2(20),
ciudad VARCHAR2(15)) ORGANIZATION INDEX;
```

**Acceso indizado
a la tabla**



Acceso a una IOT



Tablas Organizadas por Índice (IOT). Comparativa con índices normales.

Tabla Normal	Index-Organized Table
Identificador único ROWID (RID)	Identificado por la clave primaria
ROWID implícito Soporta varios índices	No tiene ROWID Soporta índices secundarios
Full Scan devuelve las tuplas desordenadas	Una recuperación completa devuelve tuplas orden. Según CP
Restricciones Unique permitidas	No soporta restricciones Unique
Soporta distribución, replicación y particionado	No soporta distribución, replicación ni particionado

Índice por Clave Invertida (reverse index). Invierte los datos del valor de la clave. Para el empleado 7698 almacena 8967. Este índice es adecuado para búsquedas basadas en predicados “=”. Con este índice se reducen los embotellamientos (retención de bloques de BD) en el índice cuando se están introduciendo los datos de forma ascendente para los valores de la clave, puesto que todos irían a la misma entrada de índice.

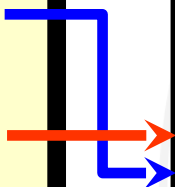
```
CREATE INDEX pie_codpie_rv_idx ON pieza(codpie) REVERSE
```

Índice sobre EMP(EMPNO)

KEY	ROWID
EMPNO	(BLOCK# ROW# FILE#)
1257	0000000F.0002.0001
2877	0000000F.0006.0001
4567	0000000F.0004.0001
6657	0000000F.0003.0001
8967	0000000F.0005.0001
9637	0000000F.0001.0001
9947	0000000F.0000.0001
...	...
...	...

Índice Invertido sobre EMP(EMPNO)

EMPNO	ENAME	JOB	...
7499	ALLEN	SALESMAN	
7369	SMITH	CLERK	
7521	WARD	SALESMAN	...
7566	JONES	MANAGER	
7654	MARTIN	SALESMAN	
7698	BLAKE	MANAGER	
7782	CLARK	MANAGER	
...
...



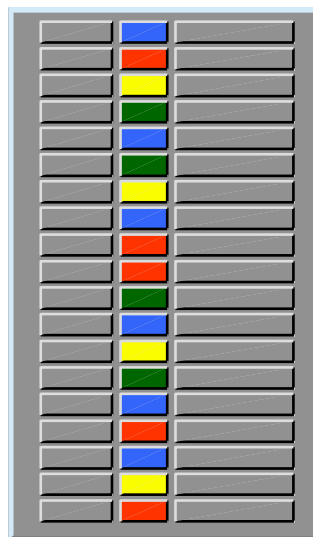
1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



Para cada valor que toma la clave almacena una secuencia de bits (tantos como tuplas contenga la tabla), el bit 1 indica que ese valor está presente en la tupla, el 0 que no lo está.

```
CREATE BITMAP INDEX pie_color_bit_idx ON pieza(color)
```

Tabla



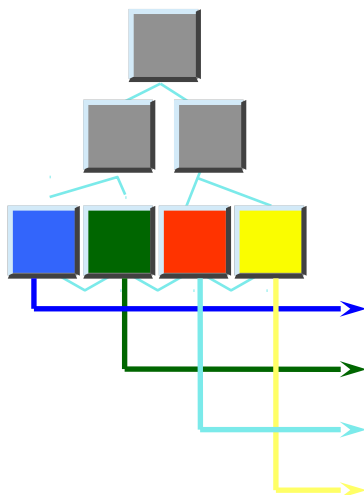
Fila 3

Bloque 10

Bloque 11

Bloque 12

Índice



Clave	ROWID inicial	ROWID final	bitmap
<Azul,	10.0.3,	12.8.3,	1000100100010010100>
<Verde,	10.0.3,	12.8.3,	0001010000100100000>
<Rojo,	10.0.3,	12.8.3,	0100000011000001001>
<Amarillo,	10.0.3,	12.8.3,	0010001000001000010>

Comparativa índices Bitmap con índices basados en B-tree.

B-tree	Bitmap
Adecuado para columnas que tomen muchos valores	Adecuado para columnas que tomen pocos valores
Actualizaciones sobre las claves no muy costosa	Actualizaciones sobre las claves muy costosa
Ineficiente para consultas usando predicados OR	Eficiente para consultas usando predicados OR
Útil para OLTP	Útil para DSS

1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



Otra forma de acceder a un registro almacenado:

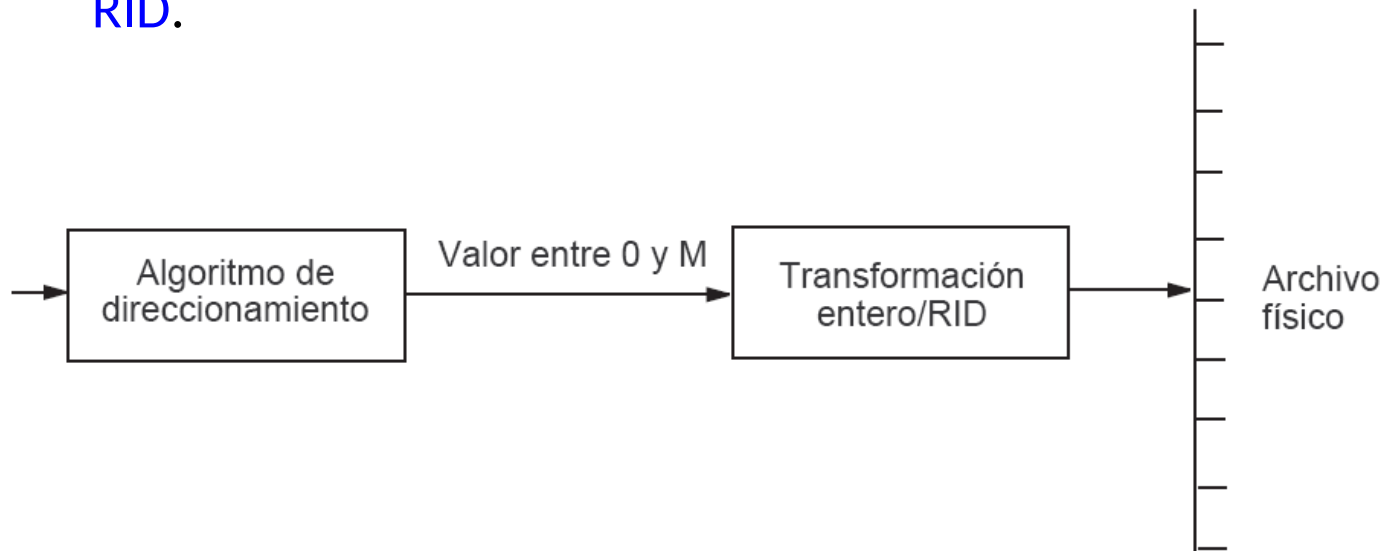
- No hay una *estructura adicional*.
- Se usa un *algoritmo* que nos *indique directamente* la *posición* del *registro* deseado.

Acceso directo:

- Calcular *directamente* la *dirección* de un *registro mediante* la *aplicación*, a un *campo* determinado del mismo, de algún *algoritmo o función*.
 - El campo debe identificar unívocamente al registro.

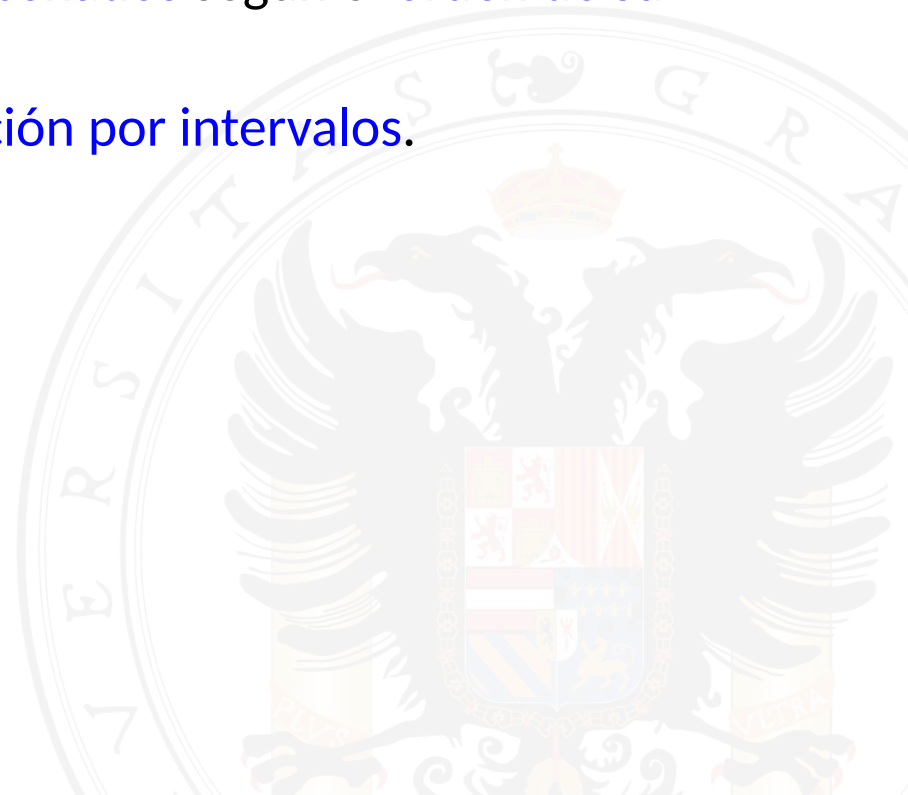
Funcionamiento:

- Normalmente no es posible establecer una **clave física** que sea totalmente correlativa y **única** para **cada registro**.
- Hay que buscar un **algoritmo** que **transforme** los **valores** de un cierto **campo** en una **dirección**.
 - Entrada: campo clave
 - Salida: valor **entero** positivo fácilmente **transformable** en **RID**.



Los algoritmos de direccionamiento no suelen mantener el orden de la clave.

- Los registros no están almacenados según el orden de su clave física.
- Problemas con la recuperación por intervalos.



Hay una gran variedad de algoritmos:

- Dependেন del tipo de clave:
 - Si la **clave** es **alfanumérica**, hay que **transformarla** a un valor **numérico**.
- Suelen estar basados en un mecanismo de **generación de números pseudoaleatorios**:
 - **Cuadrados centrales**:
 - Se eleva la **clave al cuadrado** y se eligen **tantos dígitos centrales** como sea **necesario**:
 appletviewer <http://research.cs.vt.edu/AVresearch/hashing/midsquare.php>
 - **Congruencias**:
 - Se **divide** la clave **por M** y se **toma el resto**. (M suele ser primo).
 appletviewer <http://research.cs.vt.edu/AVresearch/hashing/modfunc.php>
 - **Desplazamiento**:
 - Se **superponen** adecuadamente los **dígitos binarios** de la **clave** y luego **se suman**.
 - **Conversión de base**:
 - Se **cambia base** de numeración y **se suprimen algunos dígitos** resultantes.
- Comparación del rendimiento de diversas funciones Hash:
 appletviewer <http://research.cs.vt.edu/AVresearch/hashing/hashfuncsum.php>
- Para ejecutar los applets mediante appletviewer hay que tener instalado el SDK Java y, desde una ventana de comandos ejecutar: `appletviewer <url>`

Salvo que el campo clave se diseñe para ello, es prácticamente imposible encontrar una transformación que dé un valor entero positivo en un rango de valores limitado tal que:

- No haya dos valores distintos de clave que den lugar al mismo número.
 - *Colisiones*.

Los algoritmos producen huecos:

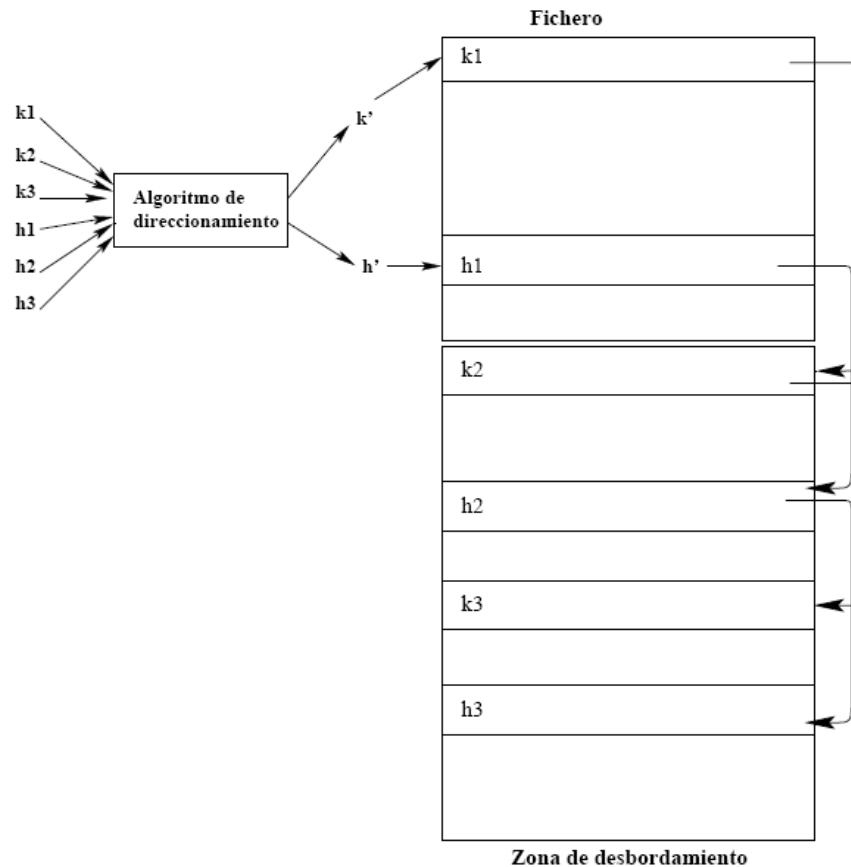
- Zonas vacías del rango de salida, no asignadas por el algoritmo.
- Se traducen en huecos en el fichero de datos.

Para gestionar colisiones y huecos:

- Combinar el acceso directo con una gestión mediante listas de colisión:
 - Zona de desbordamiento.
 - Colisión:
 - El registro problemático se almacena en la zona de desbordamiento.
 - Los sinónimos (registros con claves que producen colisión) se conectan mediante una lista.

Ejecuta:

appletviewer <http://groups.engin.umd.umich.edu/CIS/course.des/cis350/hashing/WEB/HashApplet.htm> ,
selecciona “Chaining W separate overflow” y establece 1 en: “Slots per bucket”.

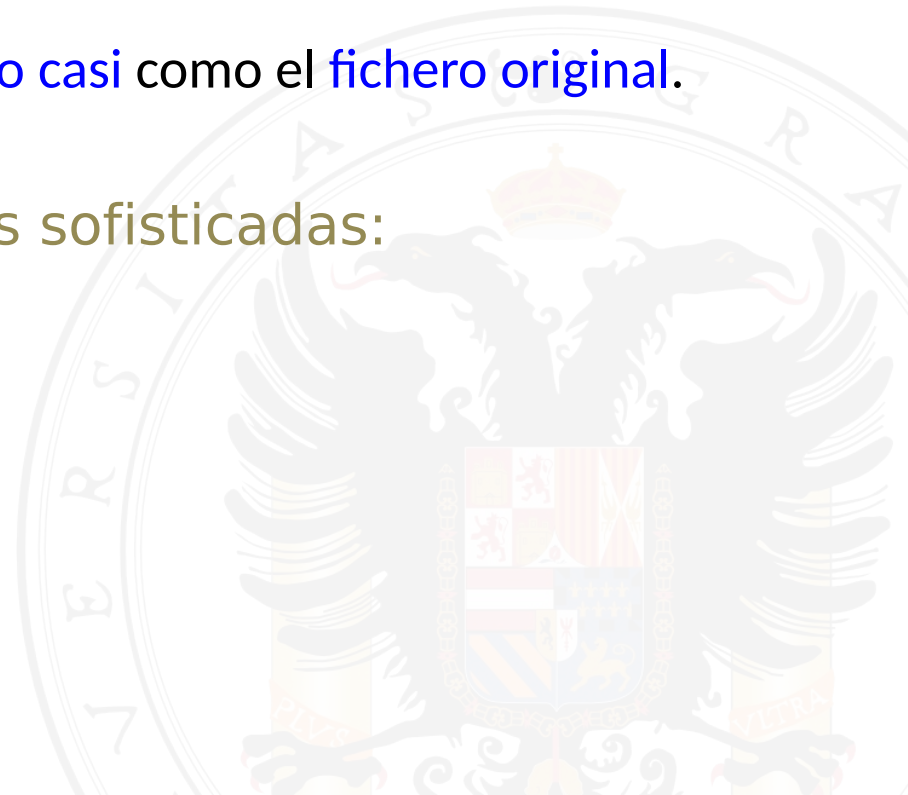


Si crecen las listas de sinónimos:

- El acceso **directo** puro **no** resulta **adecuado**.
 - **Mantener listas.**
 - Zona de **desbordamiento casi** como el **fichero original**.

Han aparecido técnicas más sofisticadas:

- **Hashing**



1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



Si el problema principal es que los valores de las claves no están uniformemente distribuidos en el intervalo $[0, M]$:

- Se acumulan en una parte de este intervalo.
- Solución:
 - Asignar más espacio a esa parte del intervalo.

Técnica:

- Se divide el espacio del fichero en “cubos” (buckets).
- El algoritmo de direccionamiento asigna cubos, no direcciones concretas.
- En cada “cubo” puede haber más de un registro.
- Ciertos rangos de valores tienen asignados más cubos que otros.
- Se complementa con el uso de “cubos de desbordamiento”.

Parámetros:

- Número de cubos.
- Tamaño de los cubos (relación con bloques físicos). (“slots”)
- La transformada clave/dirección, que debe tener en cuenta la distribución de la clave según rangos para que unos cubos no se llenen mucho y otros se queden muy vacíos.

- Selecciona “Chaining W separate overflow” en el Applet:

appletviewer <http://groups.engin.umd.umich.edu/CIS/course.des/cis350/ hashing/WEB/HashApplet.htm>

- En “Setup”:
 - Slots per bucket: 2; Buckets: 7; Size overflow: 10; Data Size: 20
- Pincha “Execute”.

Para insertar un registro:

- Transformar la clave.
- Localizar el cubo correspondiente.
- Si hay sitio se inserta el registro y hemos terminado.
- Si no hay sitio, se sitúa el registro en un cubo de desbordamiento conectándolo con el cubo que realmente le corresponde mediante punteros.
- Sobre el enlace transparencia anterior:
 - Pulsa “Step” sucesivamente para ver como se van insertando los registros de uno en uno.

El proceso de búsqueda:

- Transformar la clave.
- Localizar el cubo correspondiente.
- Realizar una búsqueda secuencial dentro del cubo.
 - Si hemos encontrado el registro, el proceso termina.
 - En caso contrario, se impone “un barrido por punteros” a través de los cubos de desbordamiento.
- Describid el proceso para encontrar un determinado valor de la clave sobre el ejemplo de la transparencia anterior.

Fichero

250	
350	
1150	
1900	
2010	
2150	
3250	
2200	
5100	
4200	
6700	
6200	
7650	

Zona de desbordamiento

1230	
0034	
0035	
3254	
2450	

1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



El hashing básico sigue teniendo problemas:

- Es necesario **conocer** la **distribución previa** de las **claves** para **asignar adecuadamente** los **cubos**.
 - En otro caso siguen apareciendo **huecos/colisiones**.
 - Al **aumentar** el número de **registros**, **aumentan** los registros en **páginas de desbordamiento**.
 - Se hacen necesarias las **reorganizaciones**.

Solución:

- Trabajar de forma **dinámica**.

Se **parte** de una configuración uniforme y de **pocos cubos**.

Los **restantes**, **se van generando** conforme se necesitan.

- Se asignan a los rangos conforme la afluencia de registros lo demanda.
- **Hashing dinámico o extensible**

Técnica:

- El valor **transformado** del campo clave nos lleva a la **entrada** de una **tabla índice** que se almacena **en memoria**.
- Allí está la **dirección del cubo donde** se **encuentran** los **registros** que tienen asociado este valor transformado.
- Puede ocurrir que **varias entradas** de la tabla conduzcan al **mismo cubo**.
- Proceso:
 - **Inicialmente**, **todas las entradas** apuntan al **mismo cubo**.
 - A medida que vamos insertando registros, **se van generando** nuevos **cubos** y **cambiando** las **salidas** de la **tabla índice**.

Algoritmo de Hashing Dinámico

- Datos de partida:
 - k = clave física para direccionar
 - $k' = h(k)$ valor entero entre 0 y M
 - n = número de bits que tiene k' en binario
 - $d \leq n$, los d primeros dígitos de k' seleccionan el cubo donde está el registro y se llaman pseudollave.
 - $b < d \leq n$, inicialmente el archivo tiene 2^b cubos distintos, como máximo tendrá 2^d

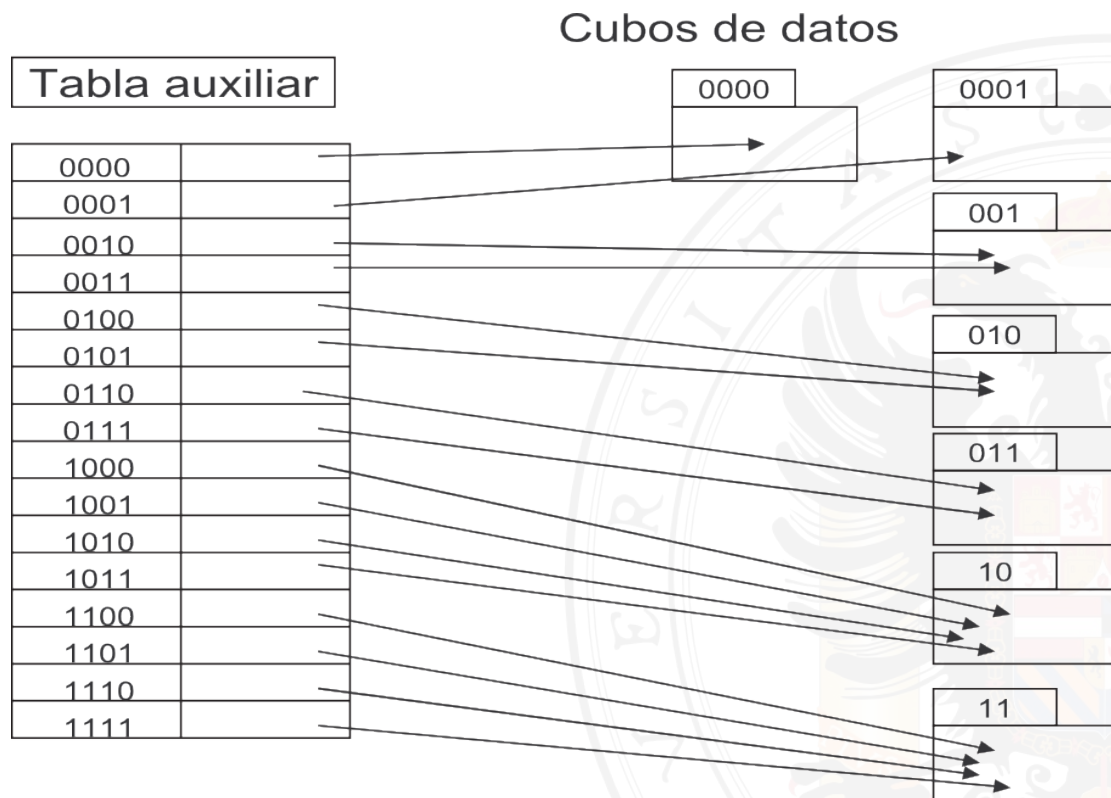
Algoritmo:

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

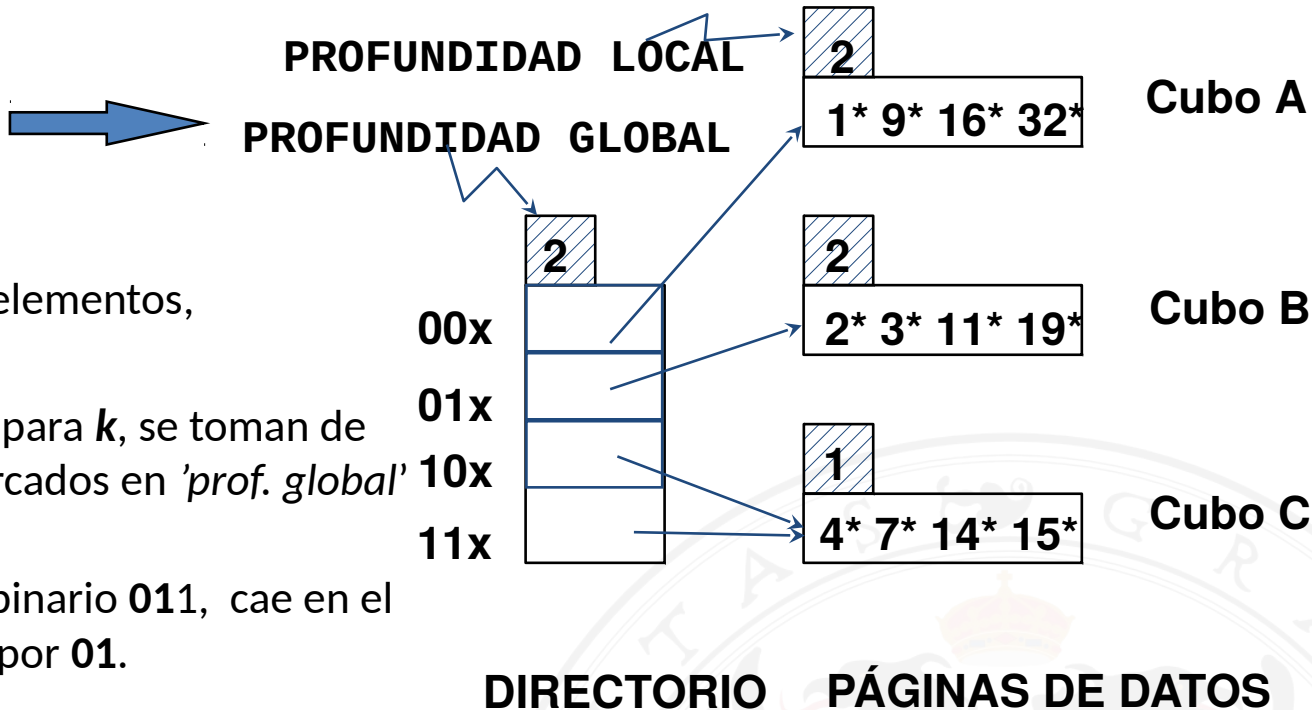
Videoanimación de Hashing Extensible

https://www.youtube.com/watch?v=8bj3wg-9E_w

Nota: En esta videoanimación el **desdoble** se realiza **partiendo** del **dígito menos significativo** del valor devuelto por $h(k)$, en vez de partir del **dígito mas significativo** como hemos **descrito** en este tema.



Ejemplo



Directorio: vector de 4 elementos,

$h(k) = k \bmod 8$

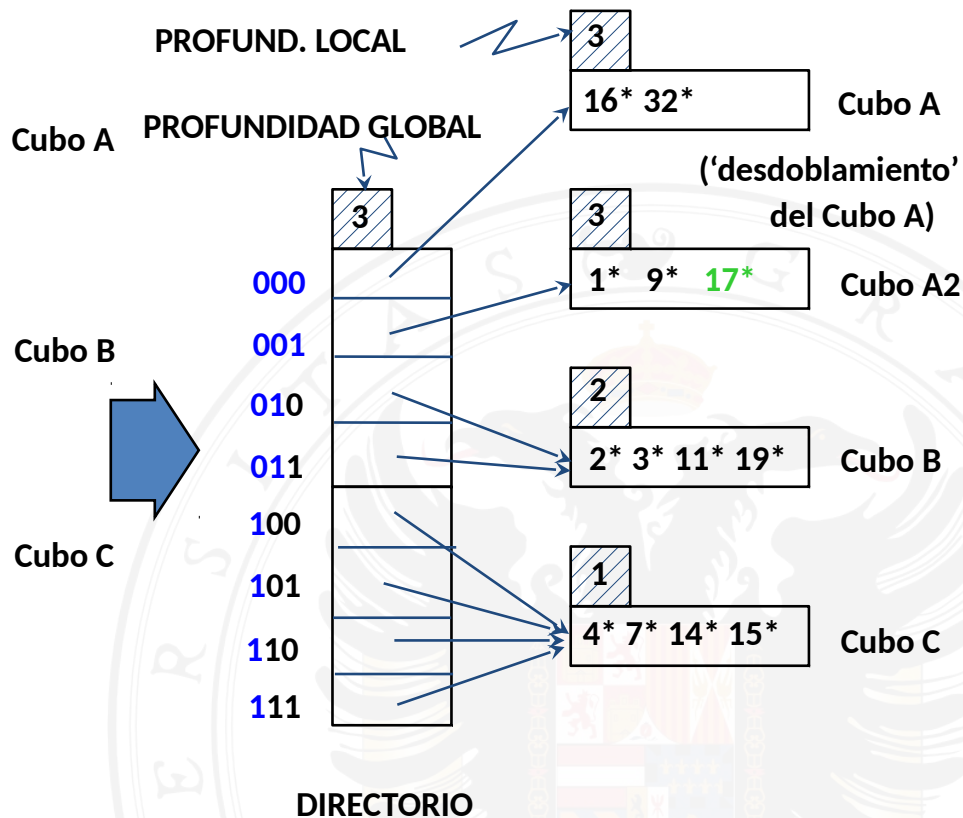
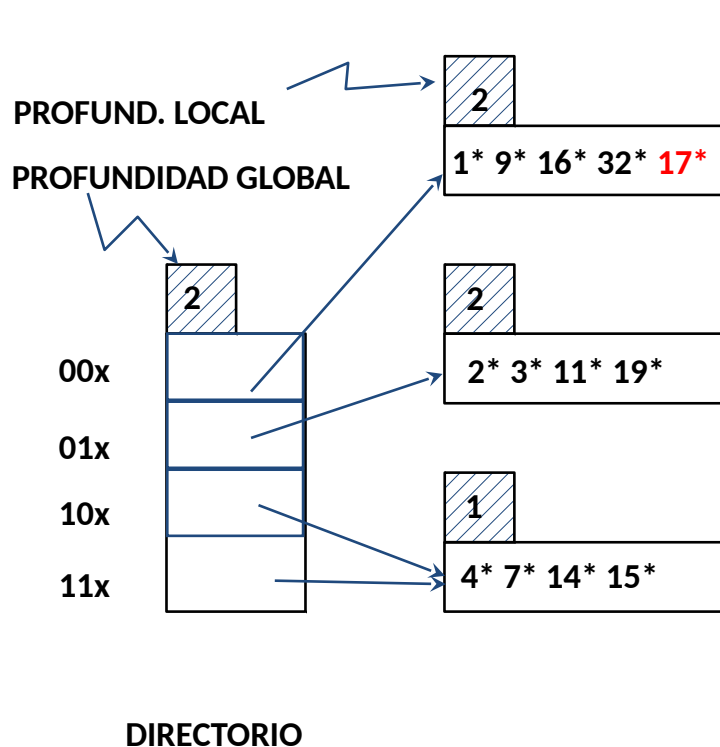
Para encontrar un cubo para k , se toman de $h(k)$ los últimos bits marcados en 'prof. global' (2).

- Si $h(k) = 3 =$ en binario **011**, cae en el cubo apuntado por **01**.

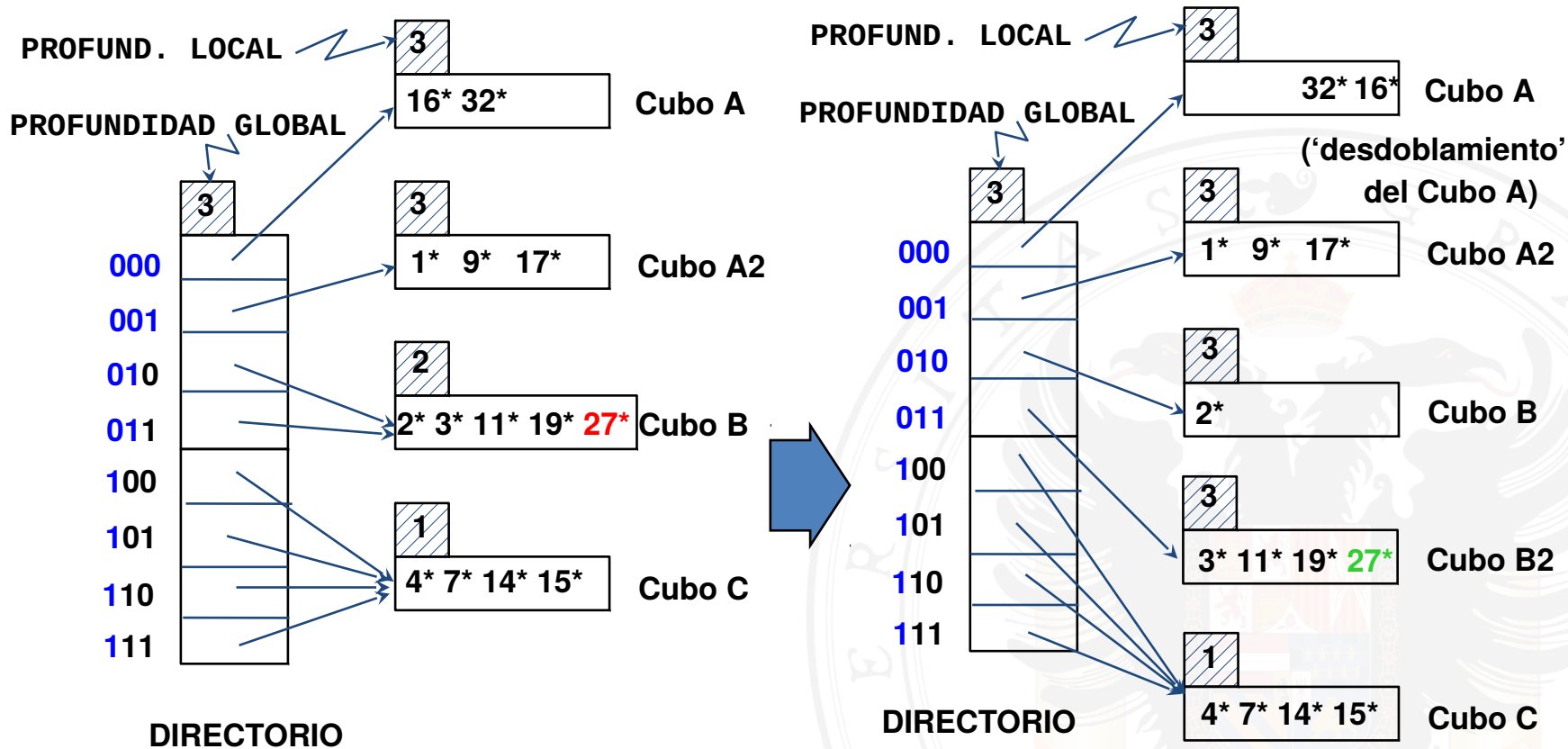
❖ **Inserción:** Si el cubo está lleno, se divide y se re-distribuyen entre los dos cubos.

❖ Si es necesario, se desdobra el directorio (no siempre es necesario) siempre que la profundidad local sea menor o igual que la global.

Insertamos **17**, $h(17)=1=001$ (Causa desdoblamiento del cubo A y del directorio)



Insertamos 27, $h(27)=3 = 011$ (Sólo causa desdoblamiento del Cubo B)



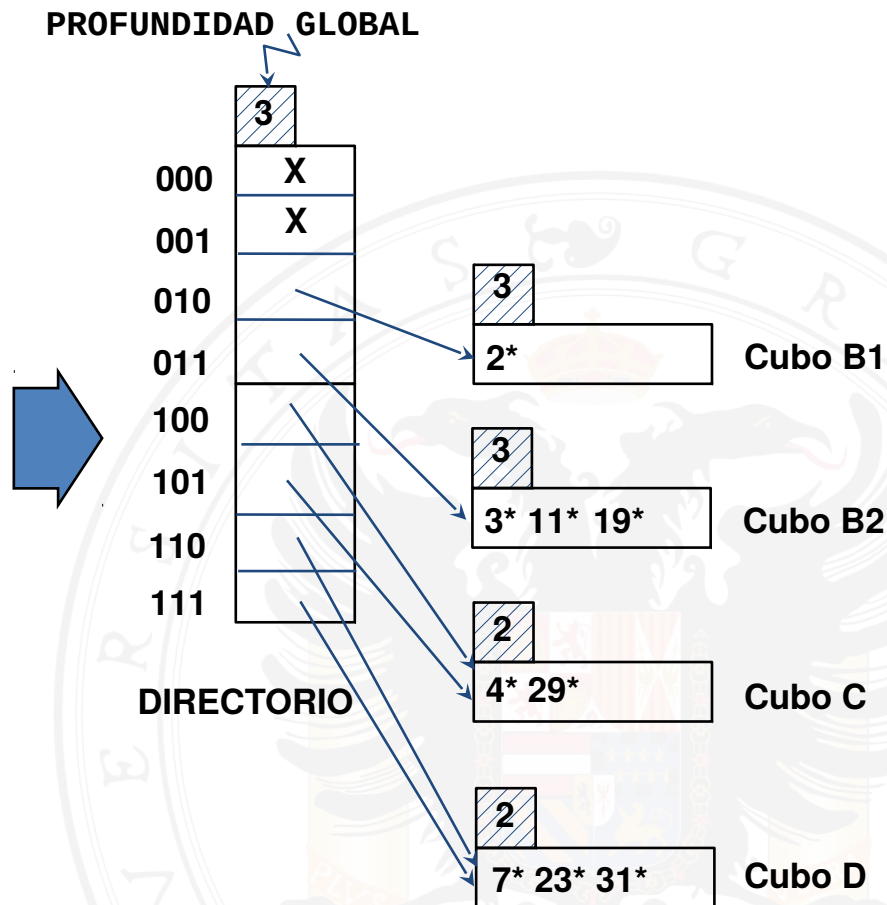
Estructura Hash. Din. para: { 2, 3, 4, 7, 11, 19, 23, 29, 31 } con Func. De Dispersión: Mod 8 y 3 registros por cubo.

Mod 8 (bin)	Nº	Prof. Local	Puntero		Mod 8 (bin)	Nº	Prof. Local	Puntero
000					000		NA	
001					001			
010	2				010	2	3	
011	3,11,19				011	3,11,19	3	
100	4				100	4,29	2	
101	29				101			
110					110	7,23,31	2	
111	7,23,31				111			



Estructura Hash. Din. para: { 2, 3, 4, 7, 11, 19, 23, 29, 31 } con Func. De Dispersión: Mod 8 y 3 registros por cubo.

Mod 8 (bin)	Nº	Prof. Local	Punter o
000		NA	X
001			X
010	2	3	CuboB1
011	3,11,19	3	CuboB2
100	4,29	2	CuboC
101			
110	7,23,31	2	CuboD
111			



Eliminación de 11 y 31

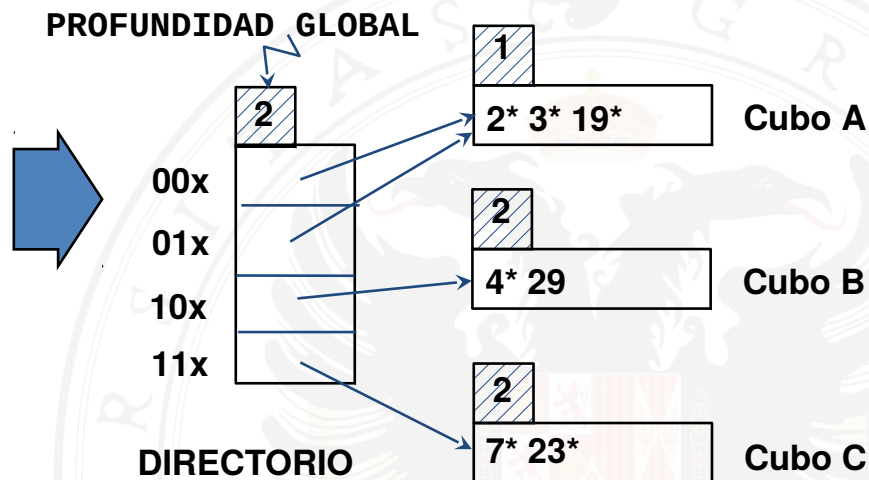
Mod 8 (bin)	Nº	Prof. Local	Puntero
000		NA	X
001			X
010	2	3	
011	3, 11 , 19	3	
100	4, 29	2	
101			
110	7, 23, 31	2	
111			



Mod 8 (bin)	Nº	Prof. Local	Puntero
000	2, 3, 19	1	
001			
010			
011			
100	4, 29	2	
101			
110	7, 23	2	
111			

Eliminación de 11 y 31

Mod 8 (bin)	Nº	Prof. Local	Puntero
000	2,3,19	1	CuboA
001			
010			
011			
100	4,29	2	CuboB
101			
110	7,23	2	CuboC
111			



Inserción de: 1, 15, 36, 40, 46 y 25

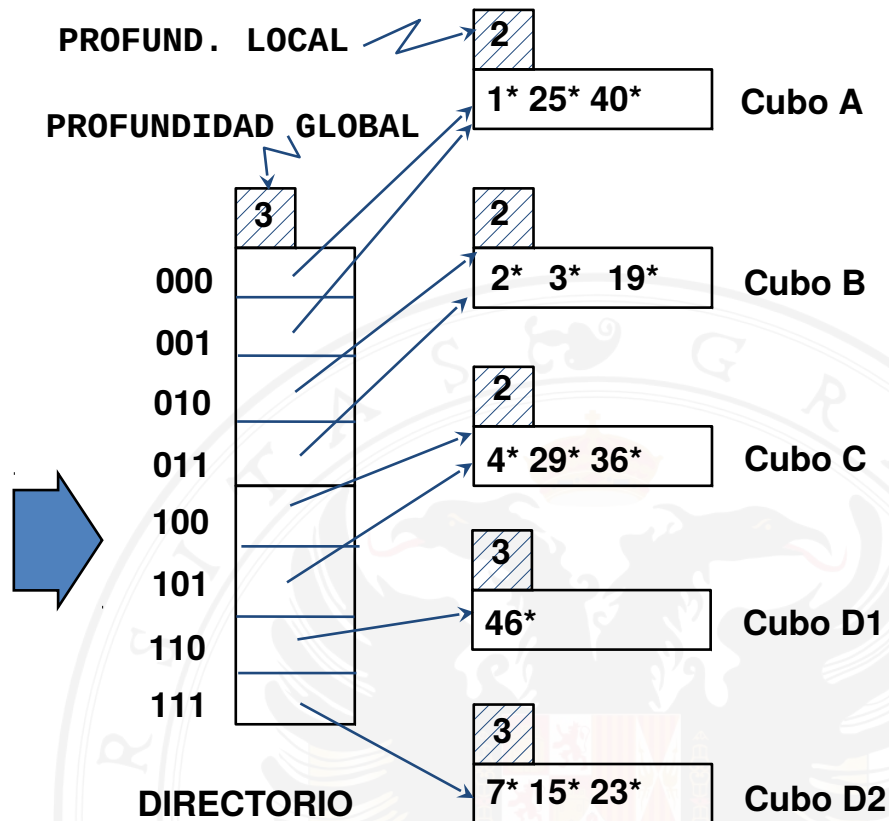
Mod 8 (bin)	Nº	Prof. Local	Puntero
000	1,2,3,	1	Cubo A
001	19,25,		
010	40		
011			
100	4,29,36	2	Cubo B
101			
110	7,15,	2	Cubo C
111	23,46		



Mod 8 (bin)	Nº	Prof. Local	Puntero
000	1,25,40	2	
001			
010	2,3,19	2	
011			
100	4,29,36	2	
101			
110	46	3	
111	7,15,23	3	

Inserción de: 1, 15, 36, 40, 46 y 25

Mod 8 (bin)	Nº	Prof. Local	Punter o
000	1,25,40	2	Cubo A
001			
010	2,3,19	2	Cubo B
011			
100	4,29,36	2	Cubo C
101			
110	46	3	CuboD1
111	7,15,23	3	CuboD2



El hashing dinámico supera los problemas clásicos del acceso directo.

También tiene sus inconvenientes:

- Utilizar una tabla índice adicional (nuevos accesos a disco si no cabe en memoria).
 - El tamaño máximo de la tabla depende de “n” y, por tanto, de la función de dispersión que se elija (cuantos valores distintos M pueda devolver).

1. Conceptos básicos
2. Dispositivos de almacenamiento
3. Métodos de acceso a la BD almacenada
4. Representación de la BD en el nivel interno
5. Organización y métodos de acceso
6. Organización secuencial
7. Indexación
8. Índices no densos
9. Índices jerárquicos
10. Árboles B+
11. Árboles B
12. Uso de Árboles B+ en BD
13. Índices BITMAP
14. Acceso directo
15. Hashing básico
16. Hashing dinámico
17. Uso del Hash en SGBD



Para usar acceso mediante Hash en un SGBD (p.e. Oracle), es preciso crear un “cluster hash” y asociar una o dos tablas a ese “cluster”.

Ej. “Cluster Hash” con una sólo tabla:

1. CREATE CLUSTER emp_hash (empno NUMBER(6)) SIZE 37 SINGLE TABLE HASHKEYS 1000;
2. CREATE TABLE employees (empno NUMBER(6) PRIMARY KEY, last_name VARCHAR2(30), department_id NUMBER(4)) CLUSTER emp_hash(empno);

La cláusula HASHKEYS indica que se trata de un clúster accedido mediante Hash.

- El valor proporcionado (1000) indica la cantidad de valores hash que se va a generar (unos 1000 códigos de empleado). Oracle busca el número primo superior a éste (1007 en este ejemplo) y esa será la cantidad de valores distintos que va generar.

Mediante la cláusula **SIZE** debemos **estimar** el tamaño que van a **ocupar** todas las **tuplas** (**empleados**) que tomen un **mismo** valor para la **clave hash** (**empno**) . En este caso hay una tupla por cada valor de la clave y su tamaño, teniendo en cuenta el de sus atributos, es **37**.

La cláusula **HASH IS** se puede usar si se cumple que la **clave** del “cluster” está compuesta por un **único atributo** de **tipo NUMBER** y contiene **enteros uniformemente distribuidos**. Esas premisas se dan en el ejemplo, por lo que se podría incluir la cláusula **HASH IS empno** en la sentencia. Esto además implicaría que **no se usaría** la **función “hash”** implementada por defecto en Oracle.

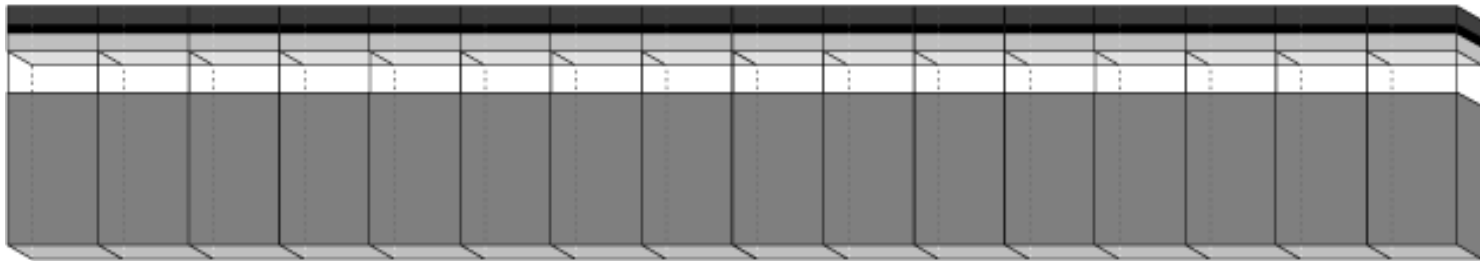
El **acceso** a un empleado mediante **Hash** a través de su código de departamento, sería:

```
SELECT * FROM employees
WHERE department_id = 20
```

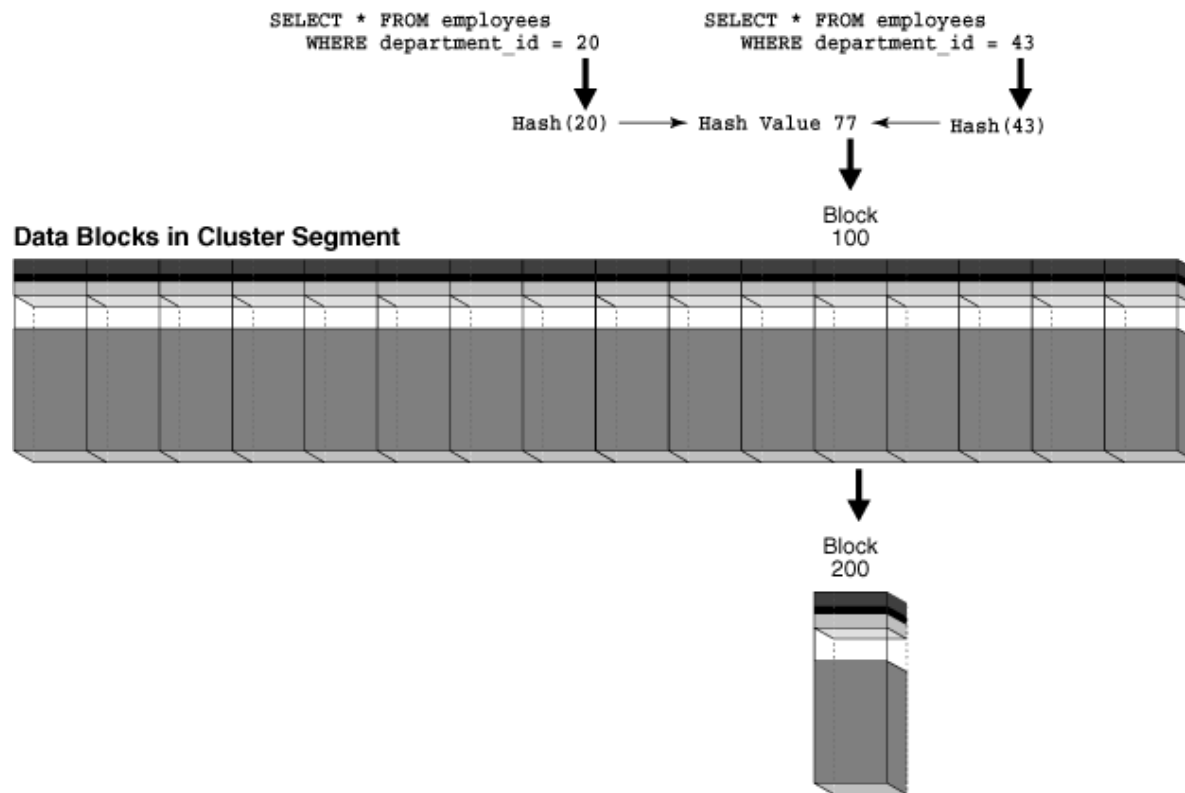
↓
Hash(20) → Hash Value 77

↓
Block
100

Data Blocks in Cluster Segment



Si al completar las inserciones de empleados se produjera un desbordamiento del bloque 100, entonces haría falta una **lectura adicional en el bloque de desbordamiento**:



Comparativa uso de Índice normal frente a Hash:

Criterio	Índice	Hash
Distribución uniforme de la clave	✖	✖
Distribución dispersa de los valores de la clave		✖
Clave infrecuentemente actualizada	✖	✖
Tablas maestro-detalle reunidas a menudo		✖
Cantidad predecible de valores para la clave		✖
Consultas usando = sobre la clave		✖