

# **t4-fbd.pdf**



**patrivc**



**Fundamentos de Bases de Datos**



**2º Grado en Ingeniería Informática**



**Escuela Técnica Superior de Ingenierías Informática y de  
Telecomunicación  
Universidad de Granada**

# Exámenes, preguntas, apuntes.

WUOLAH

Join the student revolution.

MULTI

Conéctate dónde y cómo prefieras.

Guarda tus apuntes en un lugar seguro y ordenado, y accede a ellos desde tu pc, móvil o tablet.

GET IT ON  
Google Play

Download on the  
App Store

Acceder

Registrarse

# Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



## Tema 4: Nivel interno

### 1. Conceptos básicos

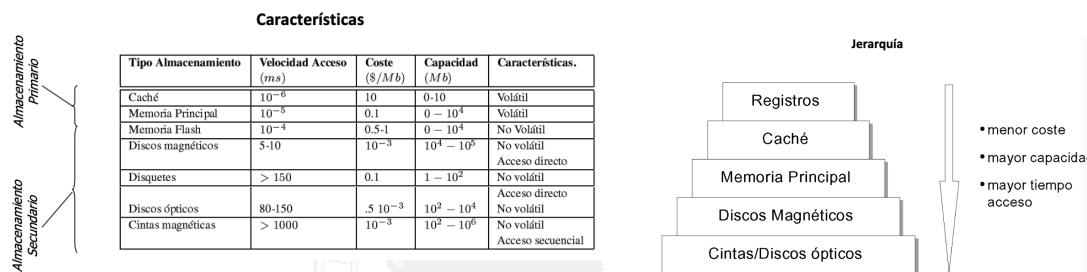
Una BD sirve para almacenar de forma permanente grandes cantidades de datos con el propósito principal de gestionar de forma eficiente los datos y su almacenamiento. Esto tiene sus consecuencias tanto en la organización lógica de los datos, como en su organización física.

**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:** está relacionado con el nivel interno. Se encuentra implementado en el SO. Proporciona al SGBD una capa de abstracción sobre el hardware. Realiza el acceso a los medios de almacenamiento mediante llamadas a servicios del sistema de archivos proporcionados por el SO.

### 2. Dispositivos de almacenamiento

Los sistemas de almacenamiento se pueden clasificar en volátil y no volátil. Los dispositivos de almacenamiento **volátil** pierden datos cuando se interrumpe la alimentación o se apagan. Por el contrario, los dispositivos **no volátiles** pueden mantener los datos independientemente del estado de la fuente de alimentación.



Dos elementos fundamentales con respecto al nivel interno: memoria principal y sistema de almacenamiento secundario

**Memoria principal:** hace trabajos de cache 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.

Es volátil, su información se pierde con las caídas del sistema. El nivel interno debe garantizar que dicha información tenga un respaldo en almacenamiento secundario para evitarlo.

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

Los datos se almacenan en bloques secundarios de almacenamiento y se recuperan para hacer las operaciones en una porción de memoria RAM.

1. El tamaño de la base de datos puede ser muy grande para almacenar la memoria
2. Si es sistema volátil la información se puede perder con una caída del sistema

Hay que encontrar el equilibrio entre los recursos disponibles y la memoria.



**Almacenamiento secundario:** se usan los SSD (discos de estado sólido) y el disco duro magnético convencional. Los sistemas se configuran en torno a las necesidades de estos discos. Hay otros tipos de almacenamiento como cintas magnéticas.

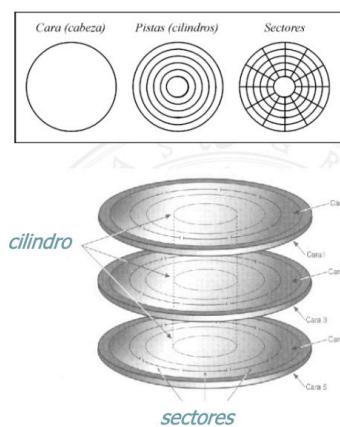
Los **discos duros** son elementos importantes en el almacenamiento secundario; son los dispositivos de almacenamiento más usado en una BD.

Se componen de un conjunto de discos magnéticos (cilindros). Cada disco tiene 2 caras en las que se guarda información y cara cada 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). Cada información se almacena en pistas, y cada pista se divide en sectores -> unidad mínima de transferencia (mínima cantidad de información que se escribe y lee).

Localización de un bloque: cilindro, superficie de disco, sector.

Parámetros que define: capacidad, tiempo medio de acceso, rpm, velocidad sostenida de lectura/escritura



### **Medidas de rendimiento:**

Tiempo medio de acceso (ta): tiempo media entre una instrucción y la obtención de información

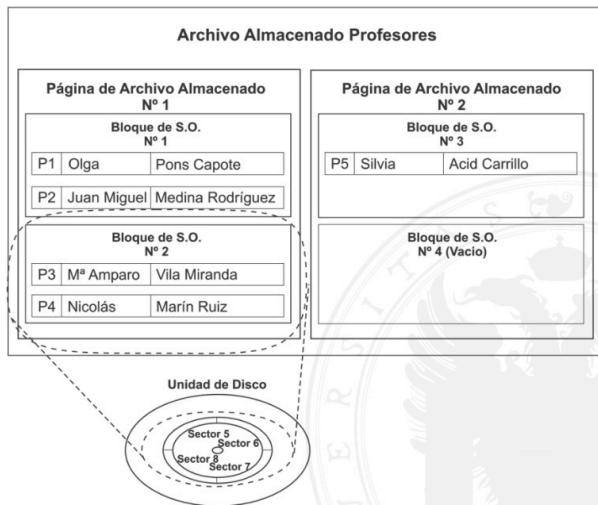
Tiempo medio de búsqueda (tb): tiempo medio de posicionamiento en la pista

Tiempo de latencia rotacional (tl): tiempo medio de posicionamiento del sector en la cabeza

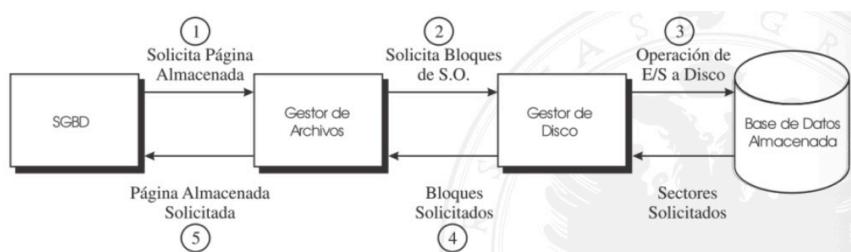
$$Ta = tb + tl$$

Tiempo medio entre fallos (MTBF)

## **3. Metodos de acceso a la BD almacenada**



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

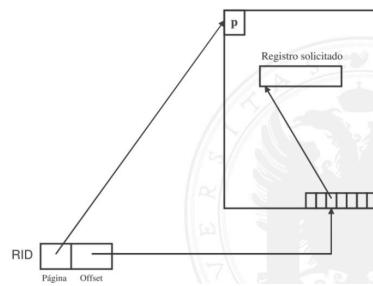


Para que el gestor de almacenamiento pueda localizar un registro, se utiliza el RID (Record IDentifier)

Cada registro almacenado se compone de:

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 SO (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

Es un módulo que forma parte del SO y que da servicio a todos los programas que se ejecutan en el SO operativo, proporcionándole la interfaz, contenido... para que pueda acceder al almacenamiento secundario y da también servicio al SGBD. Es decir, 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 (información) en conjuntos de bloques o archivos de SO. cada archivo tiene asociado un conjunto de bloques de SO para albergar la información que se almacena.

En una BD la información de datos que almacena se puede articular en uno o varios de estos archivos para almacenar su contenido.

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

Las **funciones básicas** del gestor de disco del SO son:

- **Crear un nuevo archivo de sistema operativo.** Con el correspondiente nombre que se le indique.
- **Eliminar un archivo de sistema operativo existente.**
- **Añadir un bloque nuevo al conjunto de bloques c.** Añadir más almacenamiento a un archivo del SO a través de la adición de nuevos bloques.
- **Eliminar el bloque b del conjunto de bloques c.** Ese bloque liberarlo del conjunto de bloques asociado a un archivo y ponerlo como un bloque libre para que pueda ser usado por otros archivos.
- **Devolver el bloque b del conjunto de bloques c.** Recuperar desde el almacenamiento secundario toda la información correspondiente a una página del SO.
- **Reemplazar el bloque b dentro del conjunto de bloques c.** Recuperar ese bloque y modificar la información que contiene ese bloque.

## El gestor de archivos del SGBD

El SGBD interactúa con el gestor de dichos del SO a través del gestor de archivos del SGBD.

Es un elemento del sistema que hace la transformación campos, registros y archivos almacenados a bloques y conjuntos de bloques que pueda entender el gestor de disco. Organiza los datos de manera que se minimice el tiempo de recuperación. Minimizar las E/S a disco.

Las **funciones básicas** del gestor de archivos del SGBD son:

- **Crear un nuevo archivo almacenado.** Asociar al archivo un conjunto de páginas o bloques de la BD. Se crea un archivo almacenado y se provee una cantidad de espacio para almacenar tuplas. Se crea un nuevo espacio para poder añadir tuplas.
- **Eliminar un archivo almacenado.** Los bloques que tuviera almacenado ese archivo quedan a disposición de otros archivos. (Por ejemplo: drop table).
- **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. Cada tupla se identifica un por RID que permiten encontrar el registro almacenado en cada página. Lo que se recupera es la página o bloque donde se encuentra ese registro y se ubica en la memoria para encontrar la información. (Ejemplo: select \* from ...).
- **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. Una vez que tenemos la tabla, se le manda una información al gestor de archivos y este sabe qué bloques tiene asociado. De estos bloques, el gestor sabe en cuál debe ubicar el registro. Si los bloques están llenos, recupera una página mas y ubica la información del archivo correspondiente y genera el RID del registro correspondiente. (Ejemplo: insert into table values)
- **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. Esta operación indica que vamos a borrar un determinado registro. El sistema le entrega al gestor de archivos el RID que vamos a borrar; esto le permite encontrar la página donde se encuentra ese registro, se trae la página a memoria y en memoria localiza el registro y marca el espacio que ocupa el registro como disponible y por último actualiza el correspondiente directorio.. (Ejemplo: delete from ... where ...).
- **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. Primero localizamos la pagina donde esta el registro, se la trae a memoria, localiza el registro, lo modifica si cabe la información nueva en la pagina y si se necesita mas espacio intenta meterlo en la pagina pero en otro espacio. Si no cupiera en esa página, el valor actualizado se ubicaría en otra página.

## 4. Representación de la BD en el nivel interno

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).

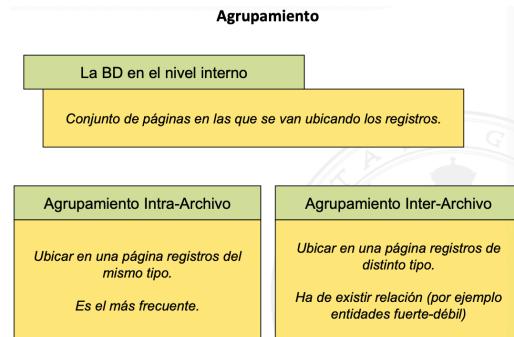
### Agrupamientos

Archivos almacenados que almacenan registros del mismo tipo -> intra-archivo

Archivos almacenados que almacenan registros de diferente tipo -> inter-archivo

# Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



## Ejemplos:

Se inserta una nueva asignatura con código A6. (Insert into)

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

0	1	2	3	4	5	24
		<b>A<sub>1</sub></b>	<b>A<sub>2</sub></b>	<b>A<sub>3</sub></b>	<b>A<sub>4</sub></b>	<b>A<sub>5</sub></b>
6	7	8	9	10	11	
	<b>P<sub>1</sub></b>	<b>P<sub>2</sub></b>	<b>P<sub>3</sub></b>	<b>P<sub>4</sub></b>	<b>P<sub>5</sub></b>	<b>P<sub>6</sub></b>
12	13	14	15	16	17	
	<b>A<sub>1</sub>P<sub>1</sub></b>	<b>A<sub>1</sub>P<sub>2</sub></b>	<b>A<sub>1</sub>P<sub>3</sub></b>	<b>A<sub>1</sub>P<sub>4</sub></b>	<b>A<sub>1</sub>P<sub>5</sub></b>	<b>A<sub>1</sub>P<sub>6</sub></b>
18	19	20	21	22	23	
	<b>A<sub>3</sub>P<sub>1</sub></b>	<b>A<sub>3</sub>P<sub>2</sub></b>	<b>A<sub>3</sub>P<sub>6</sub></b>	<b>A<sub>4</sub>P<sub>2</sub></b>	<b>A<sub>4</sub>P<sub>4</sub></b>	<b>A<sub>4</sub>P<sub>5</sub></b>
24	X	25	26	27	28	29
	<b>A<sub>6</sub></b>					

Se borra la asignatura con código A2. (Delete)

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

0	1	3	2	25	3	4	5	24
		<b>A<sub>1</sub></b>			<b>A<sub>3</sub></b>	<b>A<sub>4</sub></b>	<b>A<sub>5</sub></b>	
6	7	8	9	10	11			
	<b>P<sub>1</sub></b>	<b>P<sub>2</sub></b>	<b>P<sub>3</sub></b>	<b>P<sub>4</sub></b>	<b>P<sub>5</sub></b>	<b>P<sub>6</sub></b>		
12	13	14	15	16	17			
	<b>A<sub>1</sub>P<sub>1</sub></b>	<b>A<sub>1</sub>P<sub>2</sub></b>	<b>A<sub>1</sub>P<sub>3</sub></b>	<b>A<sub>1</sub>P<sub>4</sub></b>	<b>A<sub>1</sub>P<sub>5</sub></b>	<b>A<sub>1</sub>P<sub>6</sub></b>		
18	19	20	21	22	23			
	<b>A<sub>3</sub>P<sub>1</sub></b>	<b>A<sub>3</sub>P<sub>2</sub></b>	<b>A<sub>3</sub>P<sub>6</sub></b>	<b>A<sub>4</sub>P<sub>2</sub></b>	<b>A<sub>4</sub>P<sub>4</sub></b>	<b>A<sub>4</sub>P<sub>5</sub></b>		
24	X	25	26	27	28	29	X	
	<b>A<sub>6</sub></b>							

Se introduce un nuevo profesor con código P7. (Insert into)

1. Se ubica en la primera página libre disponible (la segunda).
2. Se ajustan las cadenas de punteros.  
La ultima pagina de profesores es la 2 (11-2). Y la página 2 ahora apunta a la siguiente página libre.

0	1	3	2	28	3	4	5	24
		<b>A<sub>1</sub></b>	<b>P<sub>7</sub></b>	<b>A<sub>3</sub></b>		<b>A<sub>4</sub></b>	<b>A<sub>5</sub></b>	
6	7	8	9	10	11	12		
	<b>P<sub>1</sub></b>	<b>P<sub>2</sub></b>	<b>P<sub>3</sub></b>	<b>P<sub>4</sub></b>	<b>P<sub>5</sub></b>	<b>P<sub>6</sub></b>		
12	13	14	15	16	17			
	<b>A<sub>1</sub>P<sub>1</sub></b>	<b>A<sub>1</sub>P<sub>2</sub></b>	<b>A<sub>1</sub>P<sub>3</sub></b>	<b>A<sub>1</sub>P<sub>4</sub></b>	<b>A<sub>1</sub>P<sub>5</sub></b>	<b>A<sub>1</sub>P<sub>6</sub></b>		
18	19	20	21	22	23			
	<b>A<sub>3</sub>P<sub>1</sub></b>	<b>A<sub>3</sub>P<sub>2</sub></b>	<b>A<sub>3</sub>P<sub>6</sub></b>	<b>A<sub>4</sub>P<sub>2</sub></b>	<b>A<sub>4</sub>P<sub>4</sub></b>	<b>A<sub>4</sub>P<sub>5</sub></b>		
24	X	25	26	27	28	29	X	
	<b>A<sub>6</sub></b>							

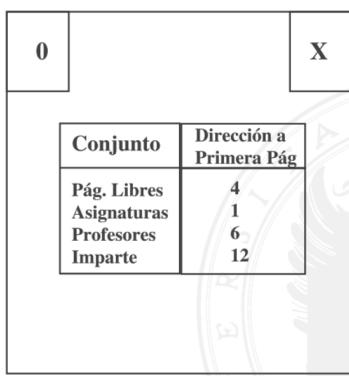


Se borra A4. (Delete)

1.Su página pasa al conjunto de páginas libres.

2. Se reorganiza la cadena de punteros de las Asignaturas. Se apunta a la siguiente página libre. La pagina 3 apunta a la 5... etc.

En cuanto a la página 0, la información que tendríamos en esa página contiene para cada archivo almacenado la página de secuencia donde comienza ese archivo:



## Ejemplo:

**Ejemplo:** Inserción de A9, borrado de A2 e inserción de A7. En realidad, las páginas contienen más de un registro.



## **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.

## 5. Organización y métodos de acceso

**Objetivo:** minimizar el número de accesos a disco  $\Rightarrow$  minimizar la cantidad de páginas de base de datos involucradas en una operación de base de datos.

Desde que solicitamos una información de unos datos hasta que los obtenemos, el tiempo tiene que ser mínimo. Es decir, que cada operación de recuperación incluya el mínimo conjunto de bloques de base de datos. Lo que buscamos es acelerar los procesos de consulta.

Ninguna de las organizaciones presentadas es mejor en términos absolutos. Los 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.

## 6. Organización secuencial

En primer lugar, la información asociada a un archivo almacenado, se organiza de forma secuencial. Esta organización provee mecanismos para que se lea el primer registro, el segundo, el cuarto... Los registros se pueden determinar. Se suelen ordenar por un campo (pero no siempre es así).

Un fichero de acceso secuencial es aquel 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).

Ejemplo en el que están ordenados por un campo: cada registro tiene su campo, el campo que utilizamos para organizar se denomina **clave física**.

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

Si lo organizamos mediante la clave física podemos hacer búsquedas por cualquier cosa, podremos acceder desde el primer al último registro, de forma secuencial.

Ejemplo: Mostrar la relación completa de departamentos. La consulta se resolvería rápidamente si los departamentos están almacenados conjuntamente en bloques contiguos de un fichero. Esto implica recorrer uno tras otro cada uno de los registros. En el peor caso (no encontrarse dicho departamento o ser el último de la lista) supone recorrer de forma completa el fichero. Búsqueda es  $O(N)$ .

Sin embargo, ¿qué pasa si queremos plantear consultas por valor de clave o por rango de valores? Lo que pasaría es que iríamos secuencialmente hasta el 20 y del 20 iría al 25. Cuando

hemos llegado al último no tenemos que seguir. 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 un registro:**

Buscar el registro.

Puede implicar una reorganización local de los registros de un bloque.

Las dos operaciones suponen:

- Escritura del bloque.
- Creación o liberación de bloques.
- 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

## **7. Indexación (Índices densos)**

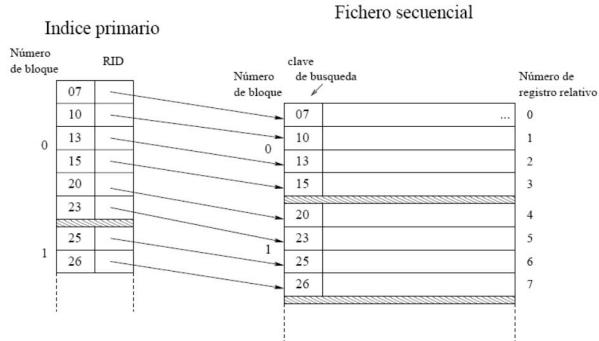
Las técnicas de índices las tenemos para acelerar las consultas. Una de las técnicas es la indexación que tiene por objeto disminuir el tiempo de acceso a los datos por una clave de búsqueda. Es similar a la idea de un índice en un libro. Existen muchas formas de llevar a cabo esta idea (ejemplo: índices glosarios).

**Ficheros indexados:** partimos de un fichero secuencial sobre el que disponemos una estructura adicional: fichero índice.

Sus registros poseen dos campos: el campo clave (la clave de búsqueda) y el campo RID (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.

La estructura sería la siguiente:



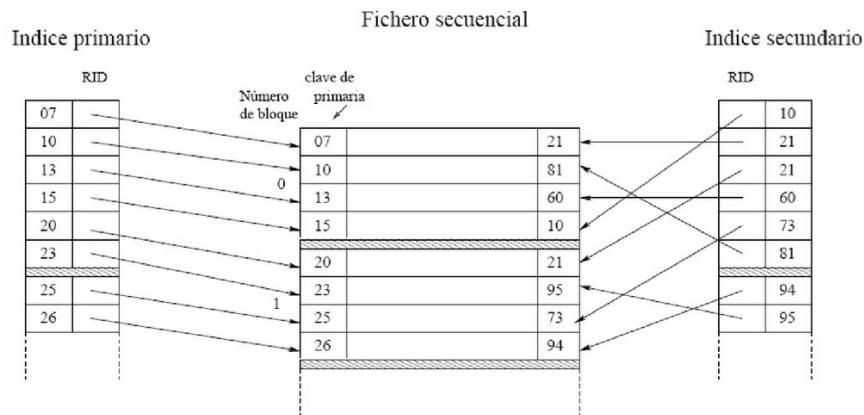
# Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



Si el índice esta creado sobre la clave física (que este ordenado de forma secuencial), se denomina **índice primario**.

Si esta construido sobre otros campos que no sean la clave física del fichero de datos, se denomina **índices secundarios**.



Proceso de consulta: hay dos tipos de consulta, que involucre la cláusula where o no la involucre.

-Cuando hacemos la consulta por un **valor de la clave**:

Sobre el índice hace una búsqueda secuencial y localizamos la clave. Obtenemos el RID del registro requerido. Volvemos al disco para recuperar el bloque de datos donde se encuentra el registro señalado por el RID. La búsqueda en el índice es más rápida.

-Cuando hacemos la consulta por un **rango de valores**:

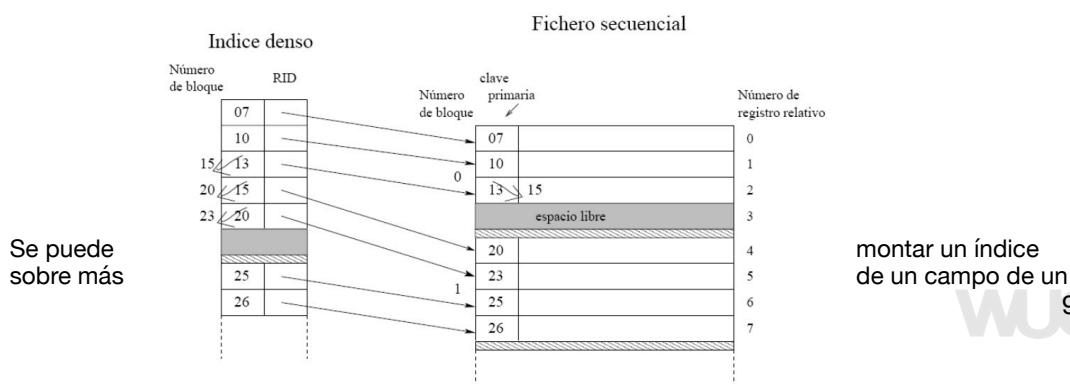
Búsqueda en el índice por valor de clave de la cota inferior. Recorrido de las entradas del índice que están en el intervalo, recuperando los registros correspondientes gracias a su RID.

Operaciones de mantenimiento:

-Inserción de un nuevo registro: buscamos a través del índice el bloque donde tenemos que insertar el registro, reorganizar las páginas y si no, llevarlo a la página de desbordamiento, es decir hay que hacer las misma operaciones que en el fichero secuencial. Hay que actualizar también el índice (inserción en un fichero secuencial).

-Borrado de un registro: lo localizamos en el archivo secuencial, marcamos el espacio que ocupa como disponible, actualizamos la secuencia y borramos la entrada en el índice. Borrado de un registro en el fichero de datos. Borrado de una entrada en el índice.

Ejemplo de borrado del registro con clave 13. (Si tuviésemos un índice secundario también habría que actualizarlo)



registro. Clave: concatenación de los campos indicados.

Hay que tener cuidado con el orden de los campos indexados. Por ejemplo, un índice sobre nombre-alumno y DNI.

Es útil para consultas que involucran:

- Nombre.
- Nombre y DNI.

No es útil para consultas sobre el DNI.

El orden si usamos un índice sobre más de un campo es relevante, ya que determina como se organiza el índice y el acceso. Y en las operaciones de consulta el índice es útil para recuperar el orden de los almacenados.

Conclusiones: a esto se le llama **índices densos**, ya que tienen un registro por cada registro almacenado que indexa. Estos índices aceleran el acceso a los datos pero ralentizan otras operaciones (ya que hay que mantener el índice). Por tanto, hay que considerar la convivencia de crear cada índice: frecuencia de consultas y frecuencia de las operaciones de mantenimiento de los datos.

Se puede montar un índice sobre más de un campo de registro. En ese caso, el valor de la clave para buscar en el índice estará compuesto por la concatenación de los campos indicados.

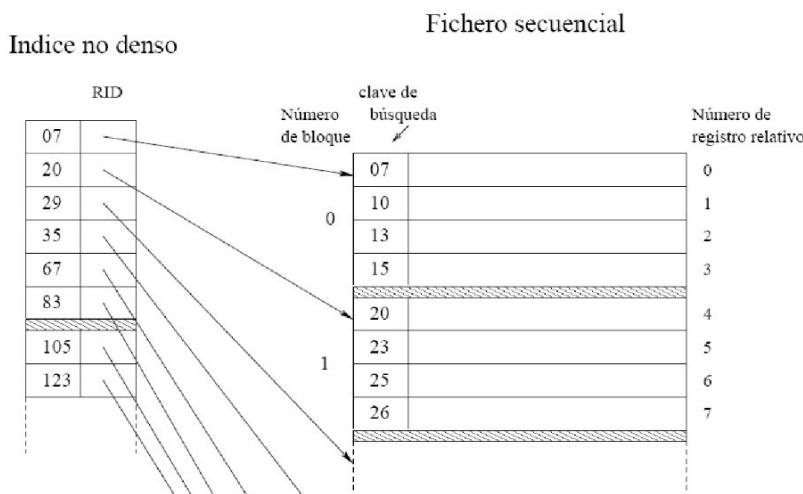
## 8. Índices no densos

Los índices son archivos almacenados. Entonces, lo ideal sería mantener el índice en memoria principal. La realidad es que los índices siguen siendo muy grandes, porque contienen todo los registros del fichero que indexan. (Estos son los índices densos)

Para acabar con lo anterior aparecen los índices no densos, que son mucho más pequeños que los densos (se reduce el tamaño) porque contienen una entrada por cada bloque secuencial que están indexando.

Están compuestos por: por la clave de búsqueda y la dirección de comienzo del bloque donde puede encontrarse el registro consultado por valor de clave.

El número de registros en el índice se reduce al número de bloques del fichero de datos. Esto hace que el acceso secuencial al índice no denso se acelere.



Un índice no denso solo se puede crear sobre una clave física. En este caso, el puntero que tiene no apunta a un registro (RID), apunta a un registro almacenado que es el primero en ese bloque. A través del índice no puedo saber si un registro está o no está, a menos que sea el del primer

bloque. En un borrado o una inserción no hay que tocar nada a no ser que afecta al primer registro del primer bloque.

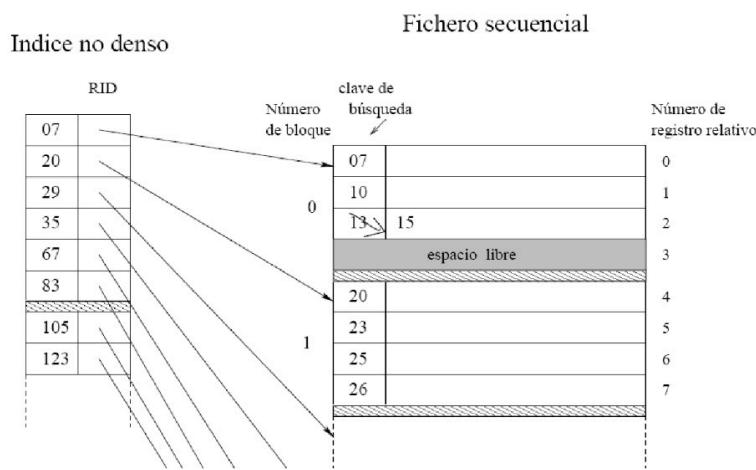
Hay dos diferencias principales entre la búsqueda de índice no denso frente al denso:

- Una vez encontrado el bloque donde podría encontrarse el registro que se está consultando, se carga el bloque en memoria y se hace una búsqueda secuencial (no tiene costes en términos de acceso a disco).
- No se tiene garantía de encontrar el registro deseado hasta consultar el bloque de datos leído.

Los indices no densos solo se pueden definir sobre la clave física (solo puede haber un indice no denso por fichero de datos).

El mantenimiento de un índice no denso es menos costoso:

- Inserción y borrado menos frecuentes.
- Solo ocurren cuando la operación afecta al valor representativo del bloque.



## 9. Índices jerárquicos

Con el objetivo de disminuir el tamaño de los indices y el tiempo necesario para recorrer el índice en busca de un registro, se planteó la posibilidad de crear índices sobre índices y varios niveles en el acceso a los datos.

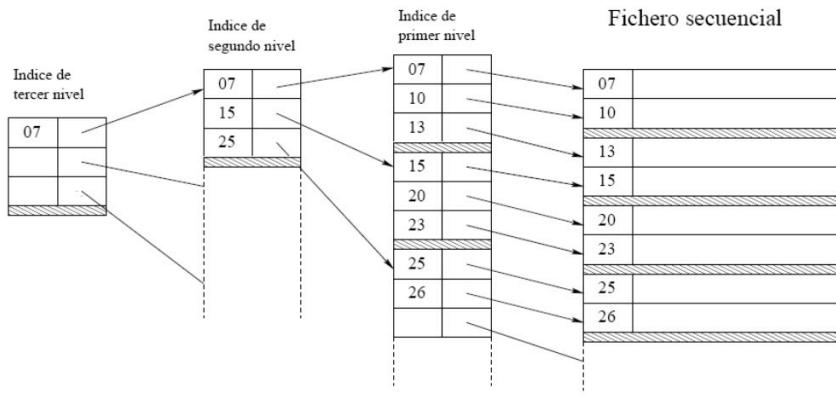
Un índice **multinivel** esta formado por:

- Un índice de primer nivel sobre el fichero de datos (que puede ser denso o no, dependiendo de la clave).
- Otros indices, no densos, construidos sucesivamente unos sobre otros (un indice de segundo nivel, que apunta a los bloques de los índices de primer nivel, y a su vez podemos tener un índice de tercer nivel, que apunta a los índices de segundo nivel... etc).

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.



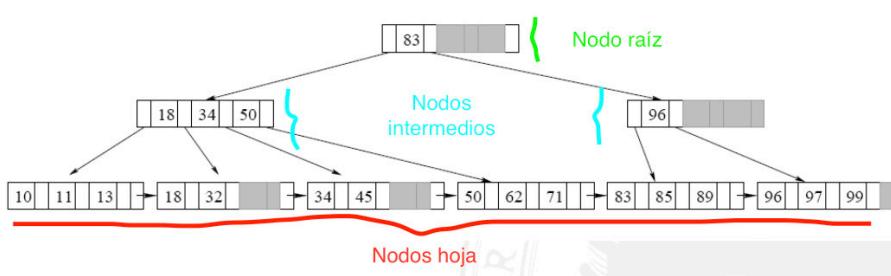
## 10. Árboles B+

Se consideran como una generalización de los índices jerárquicos (los árboles binarios balanceados) en la que los nodos pueden tener más de dos hijos. Fueron propuestos en 1972 por Bayer y McCreight.

Son equilibrados, desde el nodo raíz al nodo hoja todos los caminos tienen el mismo numero de niveles, es decir, 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 y n punteros Pi que apuntan a un nodo hijo.



Podemos observar que los valores de las claves que se almacenan en los nodos hoja están de forma ascendente; tienen también un puntero que apunta al siguiente (de forma ascendente). En el nodo raíz tenemos dos punteros que apuntan a dos subárboles y tiene como clave el 83. El subárbol de la izquierda tiene valores inferiores a 83 y el de la derecha valores superiores a 83. A

# Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



su vez, en el subárbol de la izquierda podemos ver que el primer puntero apunta a los valores del nodo hoja que son inferiores que 18, el segundo a los que son mayores o iguales que 18, el tercero a los mayores o iguales de 34... etc.

#### Restricciones dentro de los nodos:

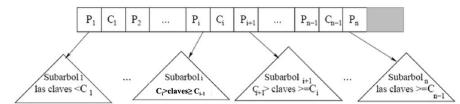
-Los valores de clave  $C_i$  están ordenados dentro del nodo.

$C_{i-1} \leq x < C_i$

Excepto para:

$i = 1$ , donde  $x < C_1$

$i = m$ , donde  $x \leq C_m$



Los nodos hoja contienen todos los valores de la clave y cada pareja de claves tendrá asociado el RID, tiene punteros al siguiente nodo hoja (que construye el conjunto secuencial) y 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.

#### Las restricciones en nodos hoja son:

- 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 llenos 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

Localizamos un **registro** y desde ahí navegamos desde el nodo raíz, bajando niveles, siguiendo la secuencia de punteros hasta llegar al nodo correspondiente donde este el valor de la clave. Buscamos el registro en el nodo hoja y, en su caso, recuperamos el registro del fichero de datos gracias al RID.

Consulta por **rango**: se localiza el nodo hoja que contiene el valor inferior y se recorren los nodos hoja hasta alcanzar el superior, recuperando los registros pertinentes del fichero de datos.

#### Inserción y borrado

Se localiza el registro, encontramos el bloque correspondiente y lo marcaríamos como disponible, actualizamos el árbol. Se utilizan algoritmos que garantizan que el árbol resultante sea equilibrado.

## 11. Árboles B

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.

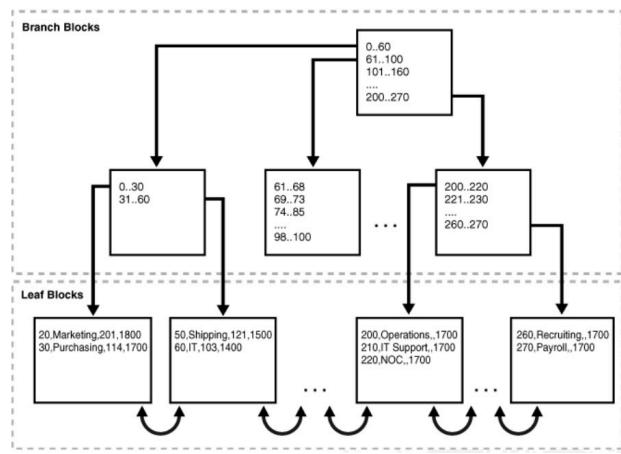
## 12. Uso de Árboles B+ en BD

- 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. (Contienen una gran cantidad de hijos)
- 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 intermedios no tenemos todos los valores de la clave, si no rangos)
  - 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.

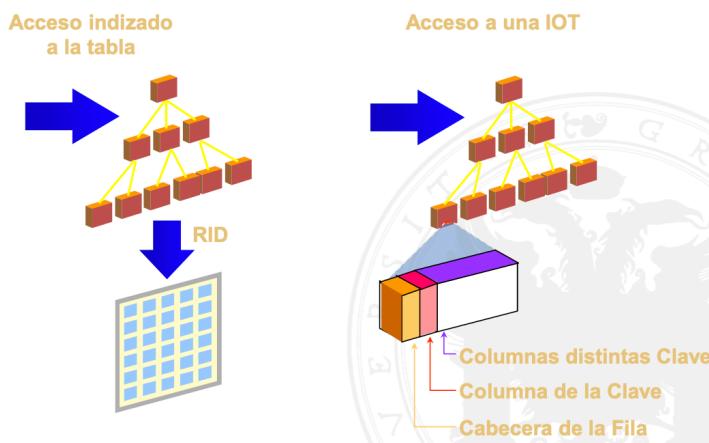
Tablas Organizadas por Índice (IOT). Las hojas contienen las tuplas en lugar del RID. Una IOT sólo puede estar organizada de esta forma mediante una clave (normalmente la clave primaria), aunque se pueden definir índices adicionales basados en otras claves. Están ordenados conforme el valor de la clave primaria y tienen asociado un árbol en las que en las hojas se almacenan el registro propiamente dicho.



Tenemos una estructura de B+. La diferencia es que los nodos intermedios se organizan por rangos. Los nodos hojas tienen un doble enlace (puntero a la siguiente secuencia y a la anterior).

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;



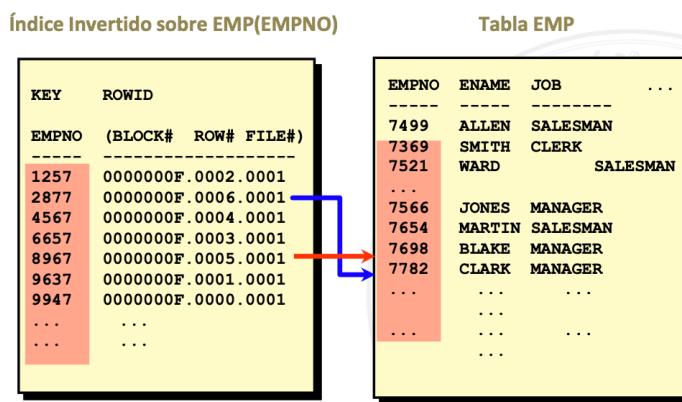
A nivel lógico es lo mismo, pero a nivel de acceso no.

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. Segun 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\_copie\_rv\_idx ON pieza(copie) REVERSE**

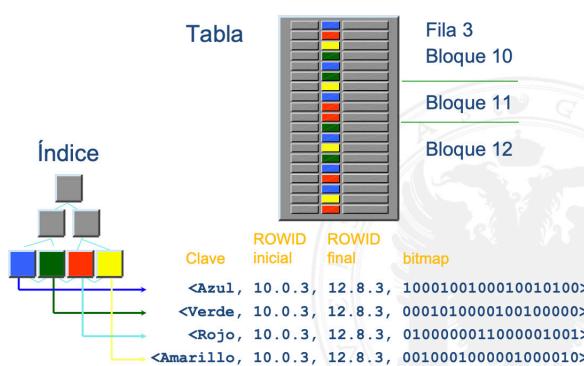


## 13. Índices BITMAP

No siguen la estructura de los arboles B+.

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. La columna que aparece coloreada es el campo sobre el que vamos a crear el BITMAP.

**CREATE BITMAP INDEX pie\_color\_bit\_idx ON pieza (color)**



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

## 14. Acceso directo

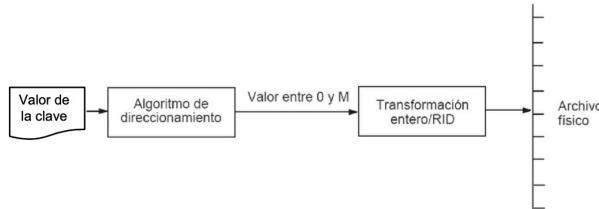
Otra forma de acceder a un registro almacenado:

- 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 de un algoritmo o función sobre un campo determinado del mismo. El campo debe identificar únicamente 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:

- Dependen 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.
  - Congruencias: Se divide la clave por M y se toma el resto. (M suele ser primo).
  - 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

# Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



- Para ejecutar los applets mediante appletviewer hay que tener instalado el SDK Java y, desde una ventana de comandos ejecutar

Los problemas que podemos encontrar son: 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 del campo clave que den lugar al mismo número. Si ocurre, tenemos dos registros que deben ocupar la misma dirección. Si pasa, decimos que el algoritmo produce sinónimos o colisiones.
- El algoritmo no produzca muchos huecos, es decir, que no se acumulen todos los valores en una determinada zona del intervalo, dejando zonas vacías del rango de salida, no asignadas por el algoritmo. Estos 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 se conectan mediante una lista.

Si crecen las listas de sinónimos:

- El acceso directo puro no resulta óptimo por:
- Mantener listas.
- Zona de desbordamiento casi como el fichero original.

## 15. Hashing básico

**Hasing basico:** en vez de la aplicación de funcionamiento asignar el registro en una ubicación determinada, el espacio del almacenamiento de los registros lo dividimos (un cubo) y en ese cubo se pueden agrupar varios registros. Así podemos ir asignando el espacio de una forma más distribuida.

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:
  1. Se divide el espacio del fichero en “cubos”
  2. El algoritmo de direccionamiento asigna cubos, no direcciones concretas.
  3. En cada “cubo” puede haber más de un registro.
  4. Ciertos rangos de valores tienen asignados más cubos que otros.
  5. Se complementa con el uso de “cubos de desbordamiento”

Los cubos de desbordamiento se usan para los registros que ya no quepan en los cubos que les corresponde. Parámetros que son necesarios:

- Número de cubos.
- Tamaño de los cubos (número de registros por cubo). Relación con el tamaño de los bloques físicos. Ideal: un acceso a disco traerá un cubo completo.



- 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.

Hay que tenerlos en cuenta a la hora de decir cuantos cubos voy a necesitar, cuantos registros voy asignar... etc.

- 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.

- El proceso de búsqueda:

Transformar la clave.

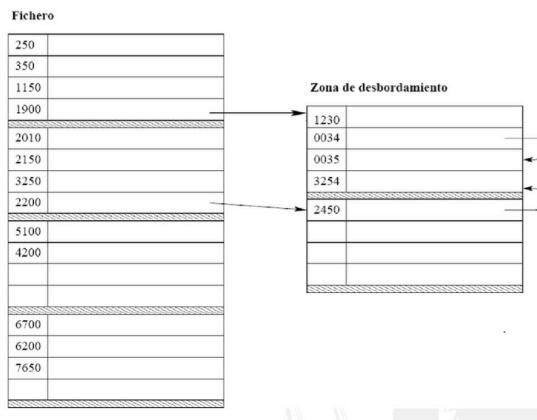
Localizar el cubo correspondiente.

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.



## 16. Hashing dinámico

El hashing básico tiene varios problemas:

- 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.

Los problemas que no resuelve el hashing básico la intenta resolver el **hashing dinámico**.

Proporciona más espacio de almacenamiento para ese rango que trabaja con más registros.

Tiene una estructura adicional. La idea es que se utiliza el valor de la clave, se utiliza una función de dispersión, y sobre el valor de la clave se obtiene un valor que se pone en binario.  $k'$  es el valor de aplicarle la función de dispersión a la clave física y es un valor entre 0 y M. Ese valor de  $k'$  lo pasamos a binario y la cantidad de bits que tiene lo vamos a llamar n. De esos bit los primeros que tomamos los denominamos d (los mas significativos) estos establecen el cubo donde está el registro al que se le ha aplicado la función de dispersión. Cuanto mayor sea, mayor cantidad de cubos tendrá.

Solución: trabajar de forma dinámica. Se parte de una configuración uniforme y de pocos cubos. Los restantes, se van generando conforme se necesiten. Se asignan a los rangos conforme la afluencia de registros lo demanda. Esta técnica se denomina: "**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  $M$  (rango de dispersión, cantidad de cubos máximos)
  - $n$ = 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  $2^d$

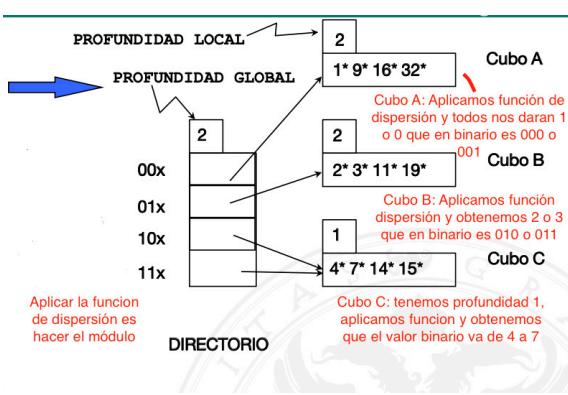
#### Algoritmo:

- 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.

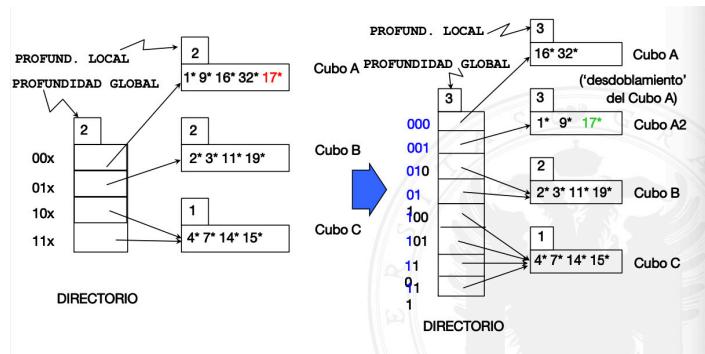
#### Ejemplos:

Directorio: vector de 4 elementos,  $h(k) = k \bmod 8$

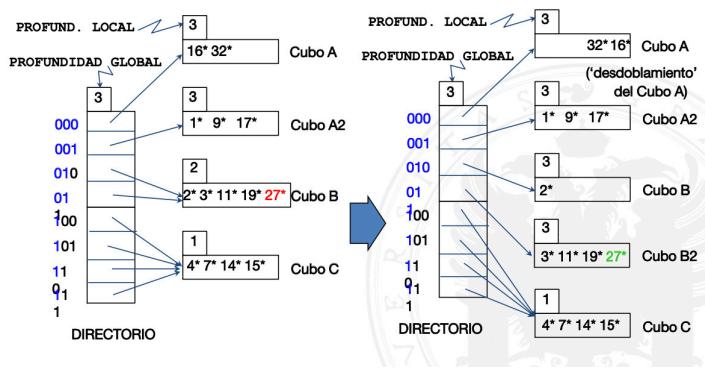
Para encontrar un cubo para  $k$ , se toman de  $h(k)$  los últimos bits marcados en 'prof. global' (2). Si  $h(k) = 3 =$  en binario 011, cae en el cubo apuntado por 01.



Insertamos 17,  $h(17)=1=001$  (Causa desdoblamiento del cubo A y del directorio). Entonces nos vamos al directorio y miramos en que cubo iría el registro. En este caso en el A, pero ya esta completo (máximo 4) entonces tenemos que dividir el cubo en otros dos cubos. En una pongo aquellos números que al aplicarle la función de dispersión me dan 000 y en otro los que me dan 001. Ahora lo que hago es acondicionar los punteros. Y ahora el directorio pasa a usar profundidad 3 (se toman los 3 bits como referencia).



Insertamos 27,  $h(27)=3 = 011$  (Sólo causa desdoblamiento del Cubo B). Este iría al cubo B, pero como esta lleno, hacemos un desdoblamiento del cubo. En el primero agrupamos los elementos que dan 010 y en el segundo los que dan 011. Actualizamos punteros.



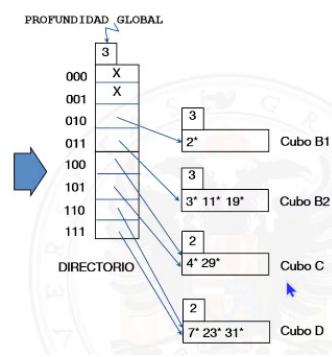
Cuidado que podemos tener desbordamiento.

Estructura Hash. Din. para: { 2, 3, 4, 7, 11, 19, 23, 29, 31 } con Función de Dispersión: Mod 8 y 3 registros por cubo. En la primera columna pongo todos los valores en binario (0-7). En la segunda pongo los números donde le corresponde (aplicando la función). Ahora intento agrupar las filas, esto se hace viendo cuales tienen el bit menos significativo diferente. La primera no tiene profundidad porque no tenemos ningún registro. La fila 010 y 011 se podrían agrupar si al juntarlas tuviese un máximo de 3 registros, pero como si las agrupo da 4, no puedo.

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

Mod 8 (bin)	Nº	Prof. Local	Puntero
000		NA	
001			
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		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			



# Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



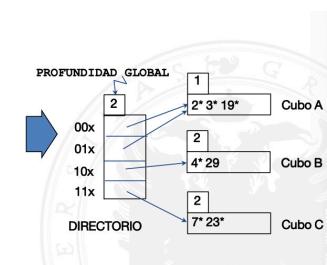
Eliminación de 11 y 31

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			

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			



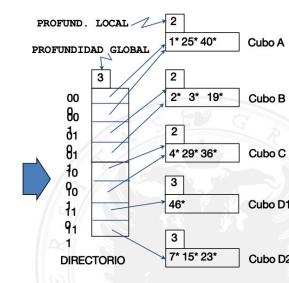
Inserción de: 1, 15, 36, 40, 46 y 25

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



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	

Mod 8 (bin)	Nº	Prof. Local	Puntero
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).

## 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 sola 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.

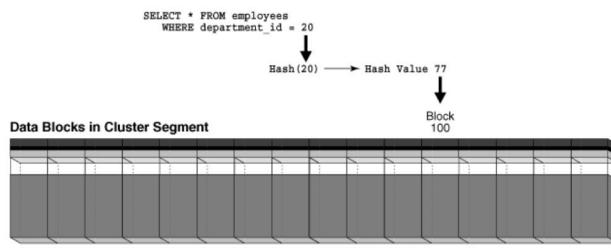
El valor proporcionado (1000) indica la cantidad de valores hash que se va a generar (unos 1000 códigos de empleado). Oracle busca el número primo superior a éste (1007 en este ejemplo) y esa será la cantidad de valores.

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 de 37 bytes.

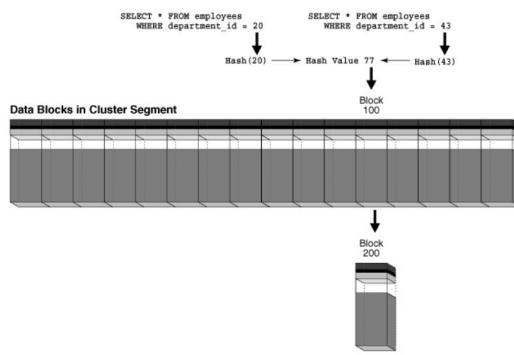
La cláusula HASH IS se puede usar si se cumple que la clave del “cluster” está compuesta por un único atributo de tipo NUMBER y contiene enteros uniformemente distribuidos. Esas premisas se dan en el ejemplo, por lo que se podría incluir la cláusula HASH IS empno en la sentencia. Esto además implicaría 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



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	<input type="checkbox"/>	<input type="checkbox"/>
Distribución dispersa de los valores de la clave		<input type="checkbox"/>
Clave infrecuentemente actualizada	<input type="checkbox"/>	<input type="checkbox"/>
Tablas maestro-detalle reunidas a menudo		<input type="checkbox"/>
Cantidad predecible de valores para la clave		<input type="checkbox"/>
Consultas usando = sobre la clave		<input type="checkbox"/>