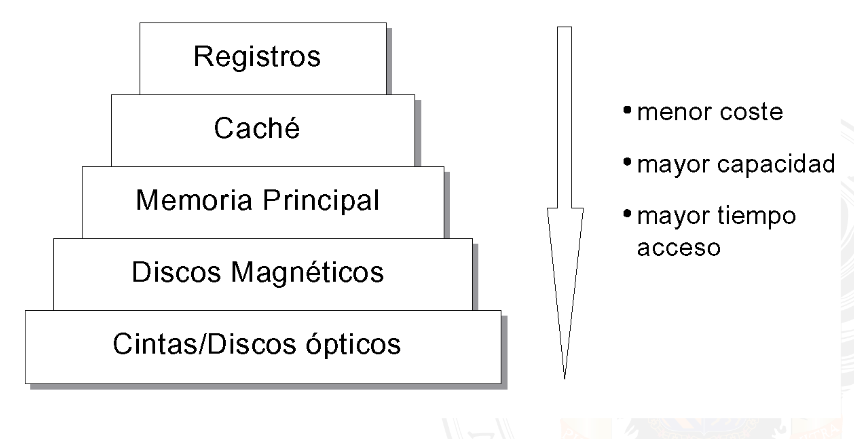
# Tema 4: Nivel Interno

## Conceptos básicos

El **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.** Además, provee al administrador de mecanismos para optimizar el almacenamiento y el acceso a los datos. Se encuentra implementado en el **SGBD.**

El **nivel físico** se encuentra implementado en el SO. Proporciona al SGBD una capa de abstracción sobre el hardware y realiza el acceso a los medios de almacenamiento mediante llamadas a los servicios del sistema de archivos proporcionados por el SO.

## Dispositivos de almacenamiento



### Memoria principal

Es el dispositivo de almacenamiento primario de los ordenadores. Se encarga de hacer trabajos de **caché** de la porción de la **BD** de uso más reciente. Es un elemento de almacenamiento intermedio que ubica de forma temporal los datos afectados por las operaciones.

Al tratarse de una memoria rápida y cara, 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.

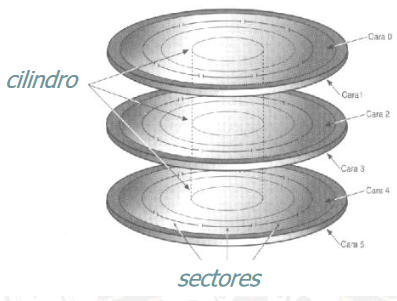
Tanto el disco duro como la memoria principal utilizan **distintos niveles de caché** para acelerar el acceso a los datos.

### Discos duros

Es el dispositivo de almacenamiento más usado en BD. Están 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). Un bloque se puede localizar en: cilindro, superficie de disco o sector.

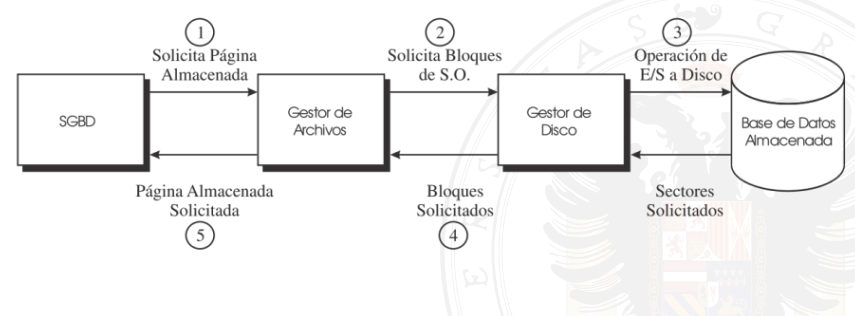
Algunos **parámetros** relacionados con los discos duros son: capacidad, tiempo medio de acceso, RPM y velocidad sostenida de lectura/escritura.



#### Medidas de rendimiento

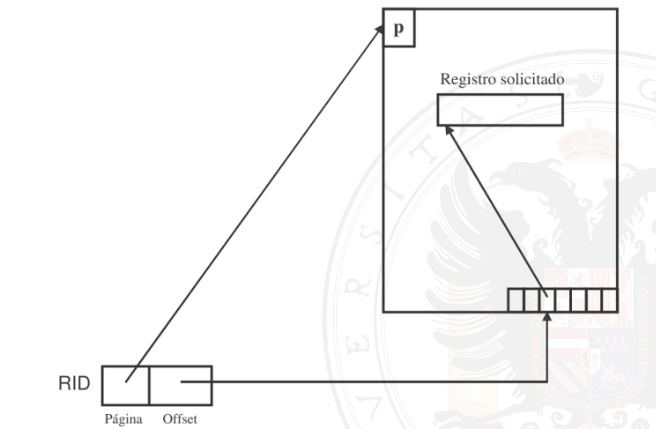
* Tiempo medio de acceso (ta): tiempo medio transcurrido entre una instrucción y la obtención de de la información.
* Tiempo medio de búsqueda (tb): tiempo medio de posicionamiento en pista.
* Tiempo de latencia rotacional (tl): tiempo medio de posicionamiento en sector.
* ta = tb + tl
* Tiempo medio entre fallos: MTBF

## Métodos de acceso a la BD almacenada



En la anterior imagen se ilustra 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 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 sistema operativo** (mínima unidad de E/S).

Para recuperar un registro almacenado hay que **determinar la página de la 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.

El gestor de disco también se encarga de gestionar el espacio libre en el disco.

#### Funciones

* **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

* 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.**
  + Solo 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 puediera, se solicita una nueva página.
  + Se devuelve al SGBD el RID nuevo.
* **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.

## 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**).

### Agrupamiento

La BD en el **nivel interno** es un conjunto de páginas en las que se van ubicando los registros.

Podemos diferenciar dos tipos de agrupamiento:

* **Agrupamiento intra-archivo:** es el más frecuente. Se basa en ubicar en una página registros del mismo tipo.
* **Agrupamiento inter-archivo:** ubicar en una página registros de distinto tipo. Ha de existir relación (por ejemplo entidades fuerte-débil).

**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.**

## Organización y métodos de acceso: Índices

El objetivo es minimizar el número de accesos a disco, es decir, **minimizar la cantidad de páginas de BD involucradas en una operación de BD.**

* Ninguna de las organizaciones presentadas anteriormente 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 en dos niveles:
  + **Organización de registros** de datos a nivel de almacenamiento.
  + **Adición de estructuras complementarias** para acelerar el acceso a dichos registros.

## Organización secuencial

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 registro suelen estar ordenados por una clave (**clave física**).

Por ejemplo, si queremos mostrar la relación completa de departamentos en una BD, la consulta se resolvería rápidamente si los departamentos están almacenados conjuntamente en bloques contiguos de un fichero.

Sin embargo, si queremos plantear consultas por valor de clave o por rango de valores la cosa no es tan rápida.

**El primer caso** implica recorrer uno tras otro cada uno de los registros. En el **peor caso** (no encontrarse el departamento o estar el último), supone recorrer de forma completa el fichero. La 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).

En cuanto a la **inserción de un nuevo registro,** se trata de buscar el bloque que le corresponde (si hay sitio se inserta el nuevo registro y en caso contrario, o bien se crea 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.

En cuanto al **borrado de un registro,** se trata de buscar el registro y eliminarlo. Puede implicar una reorganización local de los registros de un bloque.

En resumidas cuentas, estos son los pasos que toman **ambas operaciones:**

* 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.

Claramente, 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.**

## Indexación

La indexación 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 varias formas de llevar a cabo esta idea.

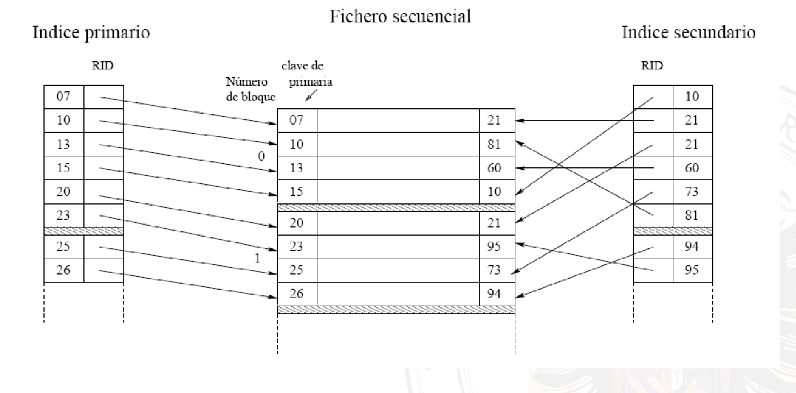
### Ficheros indexados

Partimos de un fichero secuencial sobre el que 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.



#### Proceso de consulta

Tenemos dos opciones: consulta por un valor de la clave y consulta por rango de valores.

* 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.
* 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.**

#### Inserción y borrado de un registro

En cuanto a la **inserción,** se hacen las **mismas operaciones que en el fichero secuencial.** Además, hay que actualizar también el índice (inserción en un fichero secuencial).

En cuanto al **borrado,** se trata de efectuar el borrado de un registro en el fichero de datos y el borrado de una entrada en el índice.

#### Particularidades y conclusiones

Se puede montar un índice sobre más de un campo de registro. La clave sería la **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 a **nombre o nombre y dni**, pero no es útil para consultas sobre el DNI únicamente.

Para resumir, los índices **aceleran** el acceso a los datos y **ralentizan** las otras operaciones dado que hay que mantener el **índice.**

Por tanto, hay que considerar la conveniencia de crear cada índice, según **la frecuencia de las consultas y la frecuencia de las operaciones de mantenimiento de los datos.**

## Índices no densos

Lo ideal con los índices es mantenerlos en memoria principal, pero los índices son muy grandes porque contienen todos los registros del fichero que indexa. Es decir, son **densos.**

Para reducir el tamaño, aparecen los **índices no densos:**

* Los registros están compuestos por:
  + **Clave de búsqueda.**
  + **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.

### Proceso de búsqueda

Una vez encontrado el **bloque** donde **podría** encontrarse el registro, hay que **cargarlo en memoria y hay que hacer una búsqueda secuencial** (que 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, ya que la inserción y el borrado **son menos frecuentes** y solo ocurren cuando la operación afecta al valor **representativo del bloque.**

## Índices jerárquicos

De nuevo tenemos como objetivo **disminuir** el tiempo necesario para recorrer el índice en busca de un registro: la idea es **crear índices sobre índices.** Tenemos **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** construidos 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 serían **tantos como niveles**).

El problema es que se **complica el mantenimiento del índice.**

## Árboles B+

Fueron propuestos en 1972 por Bayer y McCreight y 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 (que es el máximo número de hijos que puede tener cada nodo) es un árbol con la siguiente estructura:

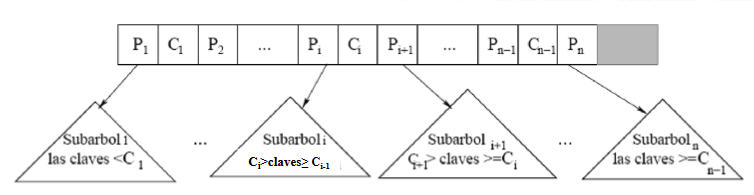
* **Nodo de nivel superior:** raíz del árbol.
* **Nodos del nivel inferior:** hojas.
* Cada nodo **distinto** de las hojas tiene como **máximo M hijos.**
* Cada nodo (**excepto raíz y hojas**) tiene como **mínimo (M+1)/2 hijos.**
* La **raíz** tiene al menos **2 hijos** si **no** es un **nodo hoja**.
* 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 Pi** que apuntan a un **nodo hijo.**

### Restricciones dentro de los nodos

Los **valores de clave Ci** están ordenados dentro del nodo.

Los **valores x** del **subárbol apuntado** por **Pi** cumplen:

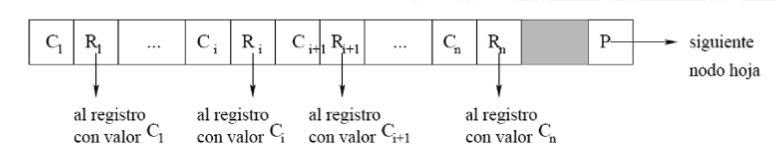
* Ci-1 <= x < Ci
* Excepto para:
  + i = 1, donde x < C1
  + i = m, donde x < Cm



### 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 secuencial**) 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. Además, **los nodos han de estar como mínimo rellenos hasta la mitad.** Todos los **nodos hoja** se encuentran al mismo nivel, es decir, es un **árbol equilibrado** y 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 al **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.

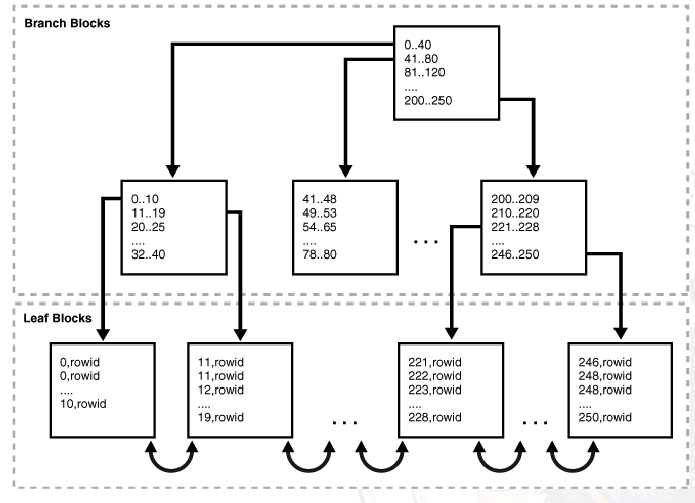
## Árboles B

Son una variante del árbol B+ en la que **no** se 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.

## Uso de árboles B+ en BD

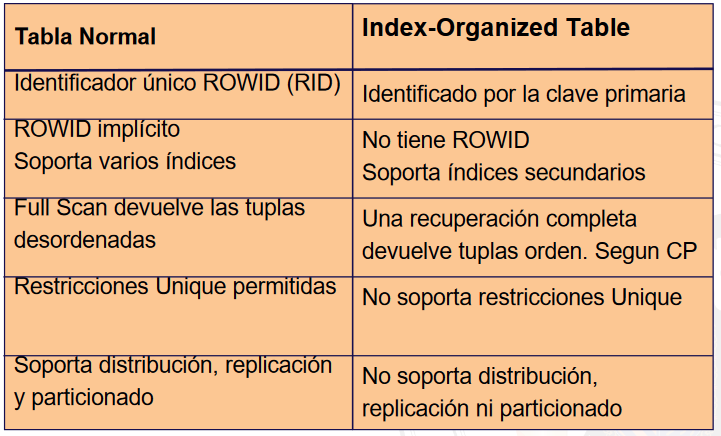
Se usan variaciones del Árbol B+, **de orden elevado,** en la que se procura que cada nodo tenga una capacidad de almacenamiento **similar** al tamaño de un **bloque de datos.** Esto reduce los accesos a disco que suelen ser los que determinan el rendimiento de las búsquedas en BD.

En los **nodos intermedios** solo están los rangos de los valores de la clave y los punteros a los nodos hijos 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. Estos nodos, 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 solo puede estar organizada de esta forma mediante una clave (normalmente la **clave primaria**), aunque se pueden definir índices adicionales basados en otras clabes.

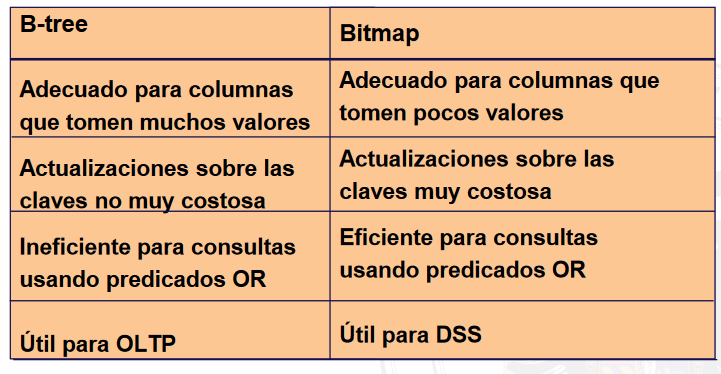


### Índice por Clave Invertida (reverse index)

Invierte los datos del valor de la clave. Para el empleado 7698 almacena, por ejemplo, 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.

## Índices BITMAP

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



## Acceso directo

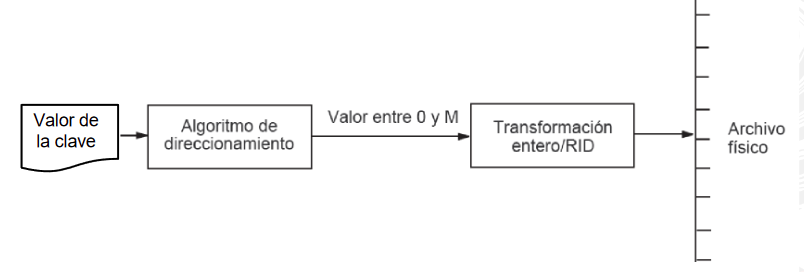
Es otra forma de acceder a un registro almacenado. No hay ninguna estructura adicional, sino que se usa un algoritmo que nos indica directamente la **posición del registro deseado.**

En resumidas cuentas, el acceso directo es calcular directamente la dirección de un registro mediante la aplicación de un algoritmo o función sobre un campo determinado del mismo. El campo debe **determinar 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.

Tendrá como entrada **el campo clave** y como salida un **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. Existen problemas con la recuperación por intervalos.

### Variedad de algoritmos

Antes de mostrar distintos ejemplos de algoritmos, cabe destacar que si la clave es **alfanumérica,** hay que transformarla a un valor numérico.

Los distintos algoritmos 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 la base de numeración y se suprimen algunos dígitos resultantes.

### Problemas

Salvo que el campo clave se diseñe para ello, es prácticamente imposible encontrar una transformación que de 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 (se producen colisiones).**

Además, los **algoritmos producen huecos,** que son zonas vacías del rango de salida, no asignadas por el algoritmo. Se traducen en **huecos en el fichero de datos.**

Se pueden 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.**

No obstante, si crecen las listas de sinónimos el acceso directo puto no resulta óptimo porque tiene que mantener las distintas listas y la zona de desbordamiento casi alcanza un tamaño como el del fichero original.

## Hashing básico

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

* Es decir, se acumulan en una parte de este intervalo.
* La solución es asignar más espacio a esa parte del intervalo.

Esta asignación de más espacio se hace así: **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 y se complementa con el uso de “cubos de desbordamiento”.**

Los parámetros necesarios para el hashing son:

* Número de cubos.
* Tamaño de los cubos (relación con bloques físicos).
* La transformación 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.

### Inserción de un registro

* Transformar la clave.
* Localizar el cubo correspondiente.
* Si hay sitio se inserta el registro y hemos terminado.
* Si no lo hay, se sitúa el registro en un cubo de desbordamiento conectándolo con el cubo que realmente le corresponde mediante punteros.

### 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.

## Hashing dinámico

El hashing básico estudiado anteriormente sigue teniendo problemas, ya que es necesario conocer la distribución previa de las claves para asignar adecuadamente los cubos. En otro caso, siguen apareciendo huecos/colisiones. Además, al aumentar el número de registros, aumentan los registros en páginas de desbordamiento. Se hacen necesarias las reorganizaciones.

La solución radica en 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.

### 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 <= n, los d primeros dígitos de k’ seleccionan el cubo donde está el registro y se llaman pseudollave.
* b < d <= n, inicialmente el archivo tiene 2^b cubos distintos, como máximo tendrá 2^d.

El algoritmo tiene la siguiente forma:

* 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.**

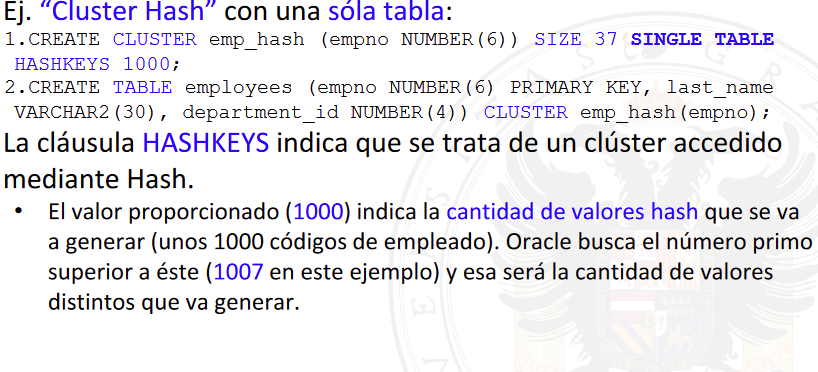
**El hashing dinámico supera los problemas clásicos del acceso directo.**

No obstante, también tiene sus inconvenientes:

* Utilizar una tabla índice adicional:
  + El tamaño de la tabla depende de **n** y por tanto, de la **función de dispersión** que se elija (cuántos valores distintos M pueda resolver).

## Uso del Hash en SGBD

Para usar acceso mediante Hash en un SGBD, es preciso **crear un “cluster hash”** y asociar una a dos tablas a ese cluster.



Mediante la cláusula SIZE debemos estimar el tamaño que van a ocupar todas la 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 clúster 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.

