Organización y manejo de archivos

www.inf.ucv.cl/ wpalma/oma

Dr. Wenceslao Palma
wenceslao.palma@ucv.cl

Indices

Indice

Un índice es una estructura de datos que toma como entrada una propiedad de los registros- generalmente el valor de uno o más atributos- y encuentra "rápidamente" los registros que posean dicha propiedad.
Un índice permite recuperar registros sin tener que recorrer en forma completa el archivo de datos.
El o los campos sobre los cuales se construye el índice se conoce como la clave de búsqueda o simplemente como clave.

Indices sobre archivos secuenciales

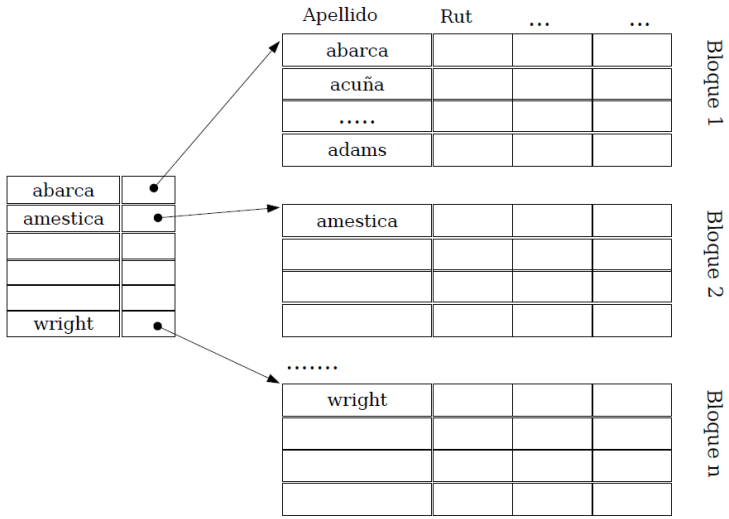
- Corresponde a la estructura de indexación más simple.
- Características:
  - El archivo de datos se encuentra ordenado y los registros son de largo fijo.
  - Cada entrada en el índice se compone de 2 campos: (a) uno del mismo tipo de la clave, (b) un puntero a bloque de disco.
  - Se conoce también como índice primario ya que la clave del índice se construye utilizando el campo clave (clave primaria) de los registros.
- El índice puede ser denso o no denso.

Indice denso

- Un índice es denso cuando existe una entrada en el índice por cada registro del archivo de datos.
- Las entradas en el índice se encuentran ordenadas de igual forma que el archivo de datos.
- Ya que las entradas en el índice ocupan menos bytes que los registros, el índice ocupará menos bloques que el archivo de datos.
- Si el índice es almacenado en forma completa en memoria principal, se puede recuperar un registro del archivo de datos con tan solo una operación de E/S.

Indice no denso

- Un índice no denso posee un número de entradas igual al número de bloques de disco.
- Cada clave contiene el valor del primer registro del bloque de datos al cual apunta.
- Sin embargo una consulta del tipo; "exite un registro con clave k?" requerirá un acceso a disco para recuperar el bloque donde se podría encontrar un registro con clave k.



Indice sobre archivos secuenciales

Operaciones

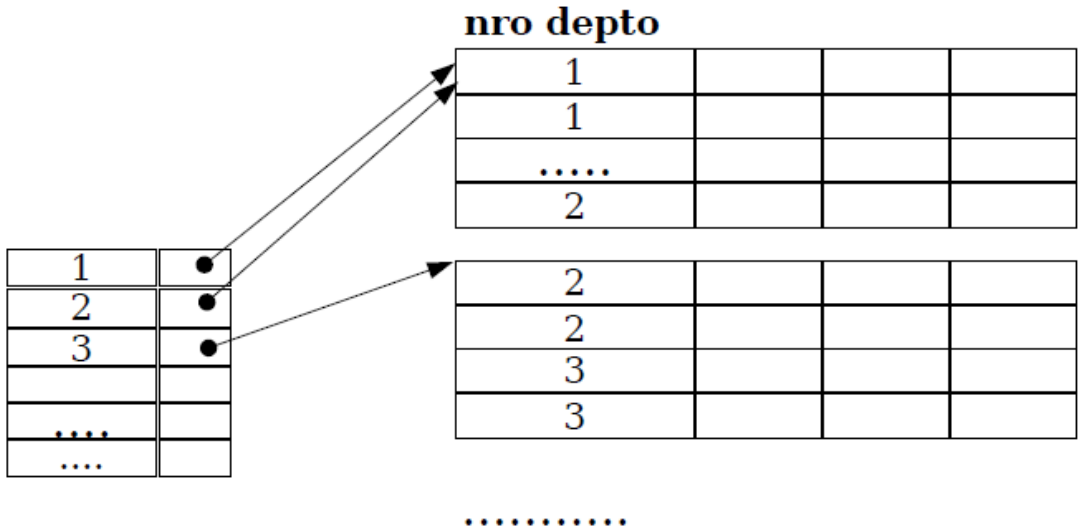
- Búsqueda: binaria en el índice.
- Inserción: corrimientos en el archivo de datos y cambios en las entradas del índice.
- Eliminación: marcas de borrado.

Ejercicio

Suponer un disco con bloques físicos de 1KB y punteros a bloque de 6 bytes. Sobre aquél se desea almacenar un archivo de 80000 registros, cada uno de los cuales tiene un tamaño de 120 bytes. Cuantos accesos se necesitan para alcanzar un registro si se contruye un índice primario no denso sobre la clave primaria (9 bytes).?

Indice con claves duplicadas en el archivo de datos

- También se conoce como Indice de grupo.
- El índice se construye sobre un campo que no es clave.
- Los datos están ordenados pero el campo de ordenamiento presenta repeticiones.
- Cada entrada en el índice se compone de: (a) uno del mismo tipo de la clave, (b) un puntero a bloque de disco.
- Para una entrada en el índice con valor k, el puntero contiene la dirección del bloque en donde se encuentra el primer registro con valor k.



Indice multinivel

- Si un índice compuesto de b bloques es muy grande para ser almacenado en memoria principal se podría realizar búsqueda binaria.
- Sin embargo lo anterior requiere $log_2 b$ lecturas.
- La solución puede ser la construcción de un índice no denso sobre el índice.

Indices sobre archivos secuenciales

Indice multinivel[1]

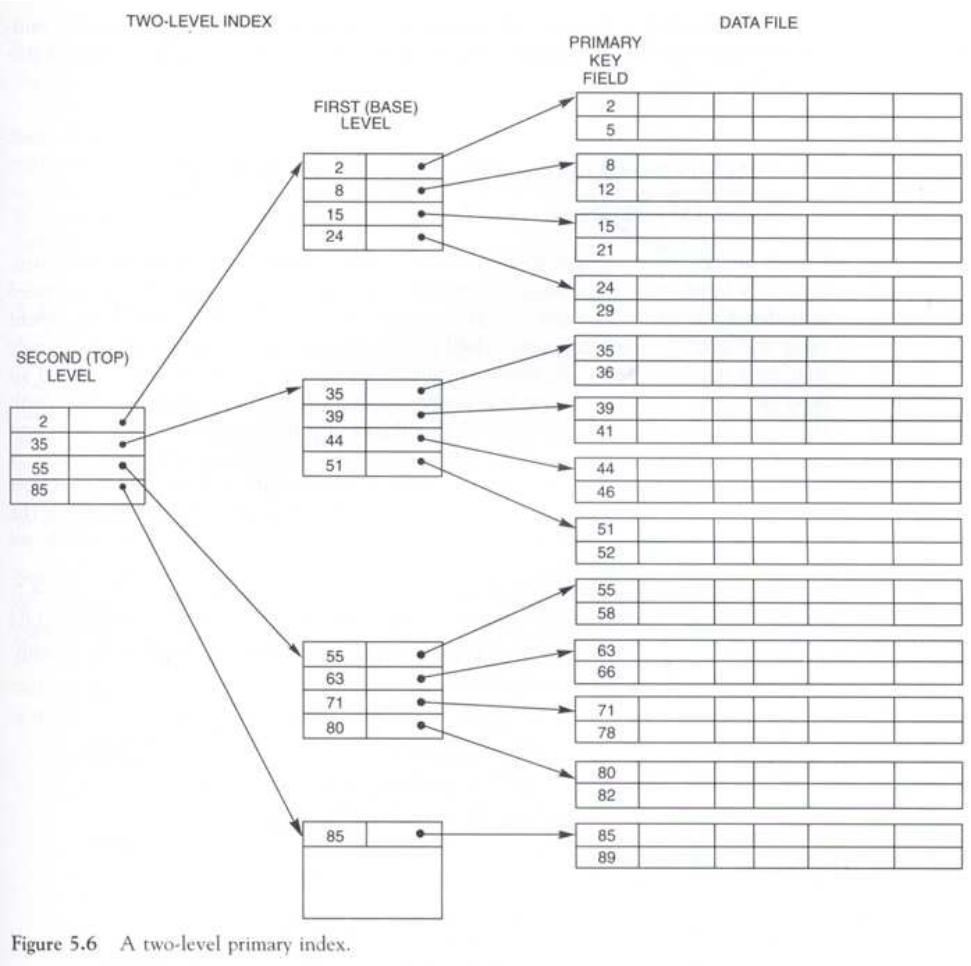


Figure 5.6 A two-level primary index.

[1]Figura extraída desde Fundamentals of Database Systems by Elmasri and Navathe, 2da edición

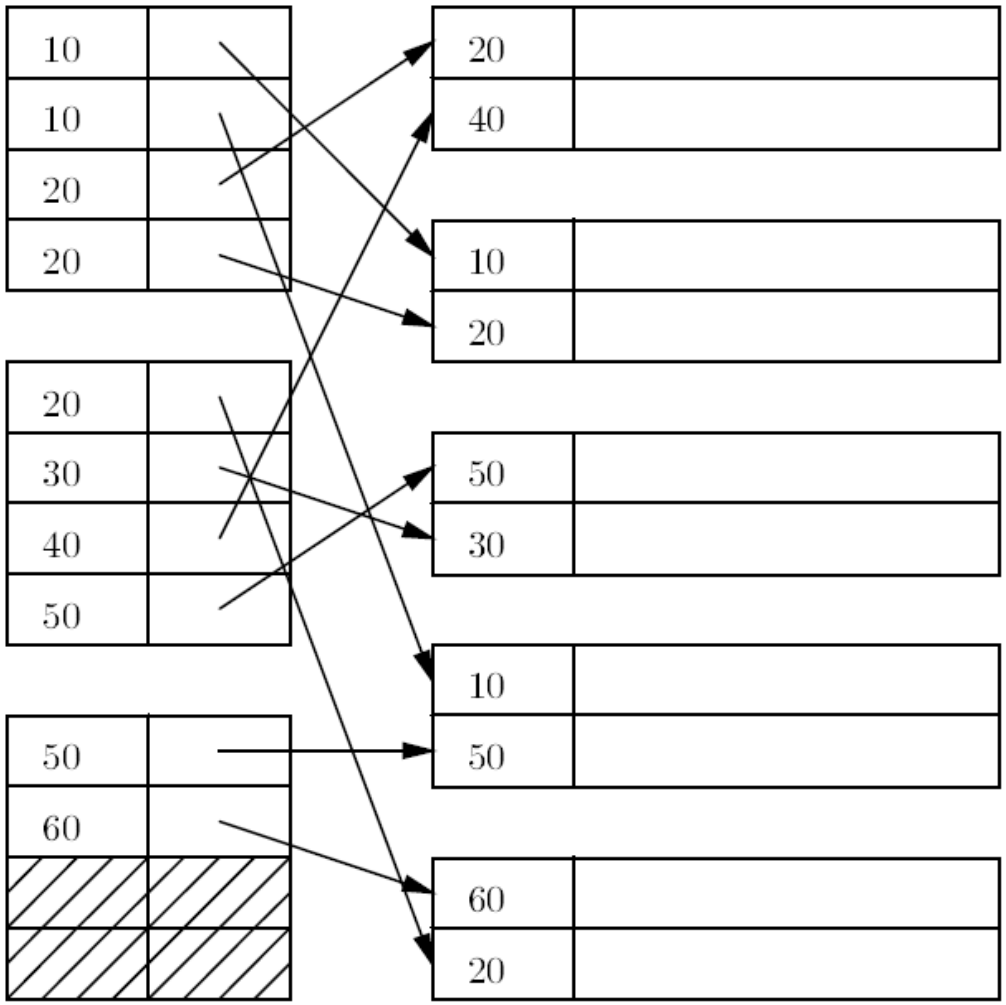
Ejercicio

Suponga un archivo de 40000 registros, c/u de 100 bytes. La clave de los registros es de 11 bytes y el puntero a bloque es de 8 bytes.
Si el índice primario no es posible almacenarlo en un disco compuesto por bloques de 1KB, construya un índice multinivel y determine cuantos accesos a disco son necesarios en cada caso.

Indice secundario

- Un índice secundario es un índice denso, generalmente con repeticiones.
- Las entradas en el índice se componen de: (1) un campo que no es clave, (2) un puntero a bloque de disco
- El índice es ordenado en base al campo de indexación.
- Para recuperar todos los registros asociados a una clave es necesario múltiples operaciones de E/S.

Indice Secundario[2]

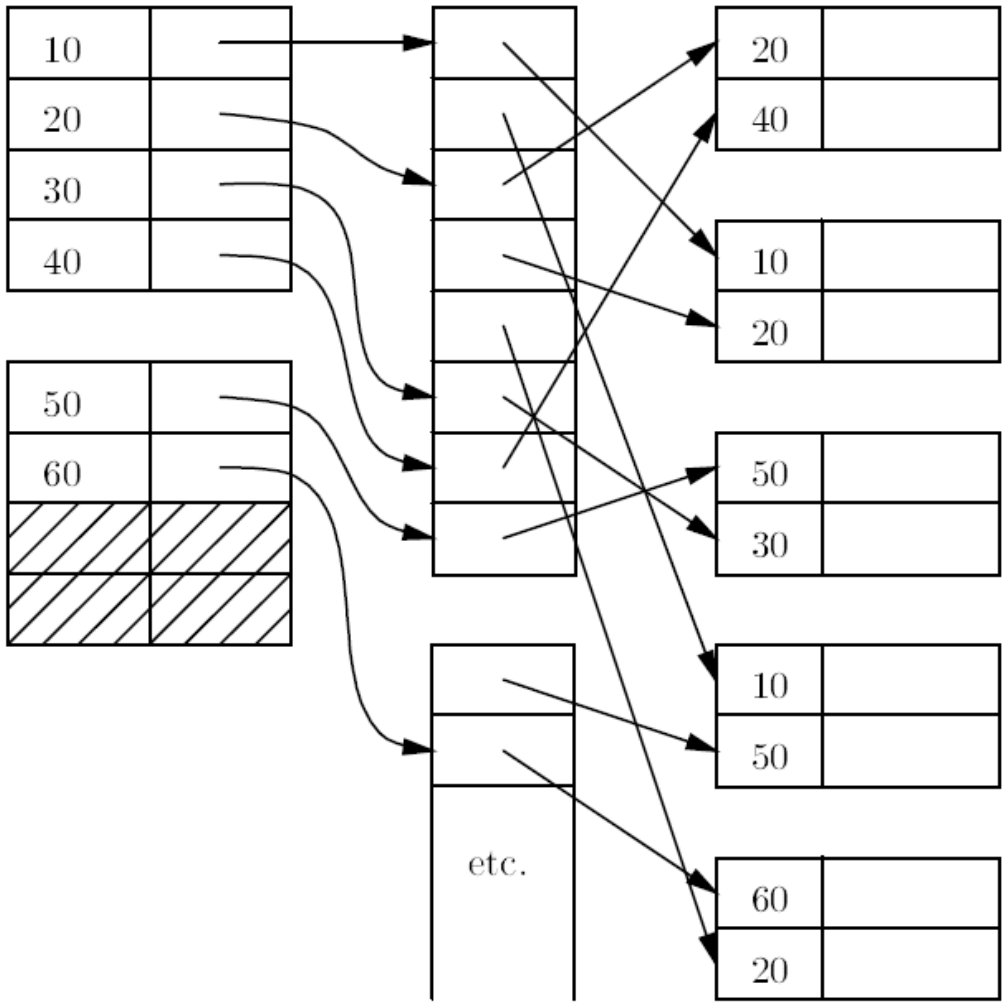


[2]Figura extraída desde Database System Implementation, Garcia-Molina, Ullman, Widom

Buckets de indirección

- Evitan las repeticiones en el índice.
- Se ubican entre el índice y el archivo de datos.
- Eventualmente esto permite ganar espacio cuando la clave ocupa más bytes que un puntero.
- Lo anterior no resulta una ventaja muy atractiva considerando que no se disminuyen las operaciones de E/S.

Indice Secundario[3]

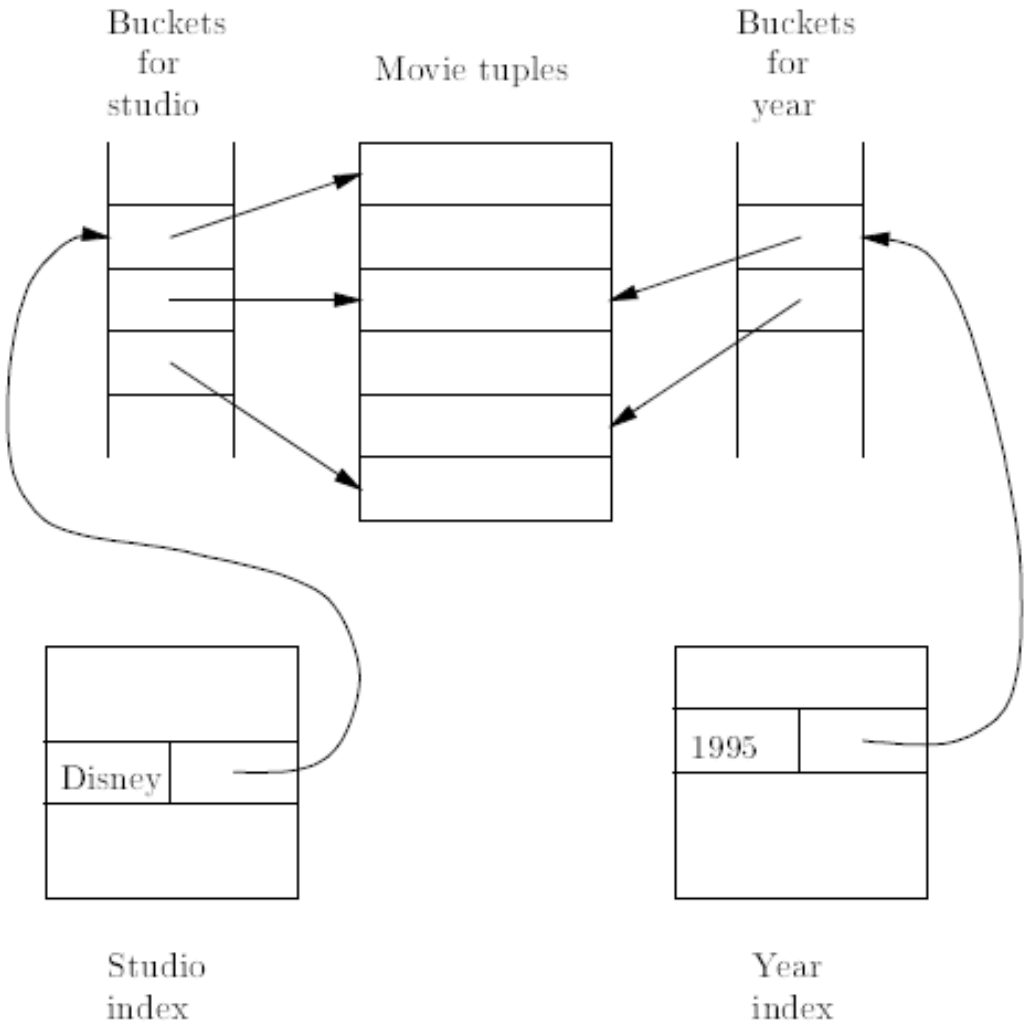


[3]Figura extraída desde Database System Implementation, Garcia-Molina, Ullman, Widom

Buckets de indirección: ventaja adicional

- Supongamos un archivo llamado Movies(title,year,length,studioName)
- Sobre este archivo existen dos índices secundarios, uno sobre **studioName** y otro sobre **year**.
- Se desea recuperar todos las películas realizadas por Disney en el año 1995.
- Usando el índice sobre **studioName** se pueden encontrar los punteros a las películas de Disney. Aún no es necesario recuperar los bloques desde el archivos de datos.
- Con el índice sobre **year** se recuperar los punteros a todas las películas realizadas el año 1995.
- Luego, se intersectan los punteros y se obtienen los registros requeridos desde el disco minimizando las operaciones de E/S :)

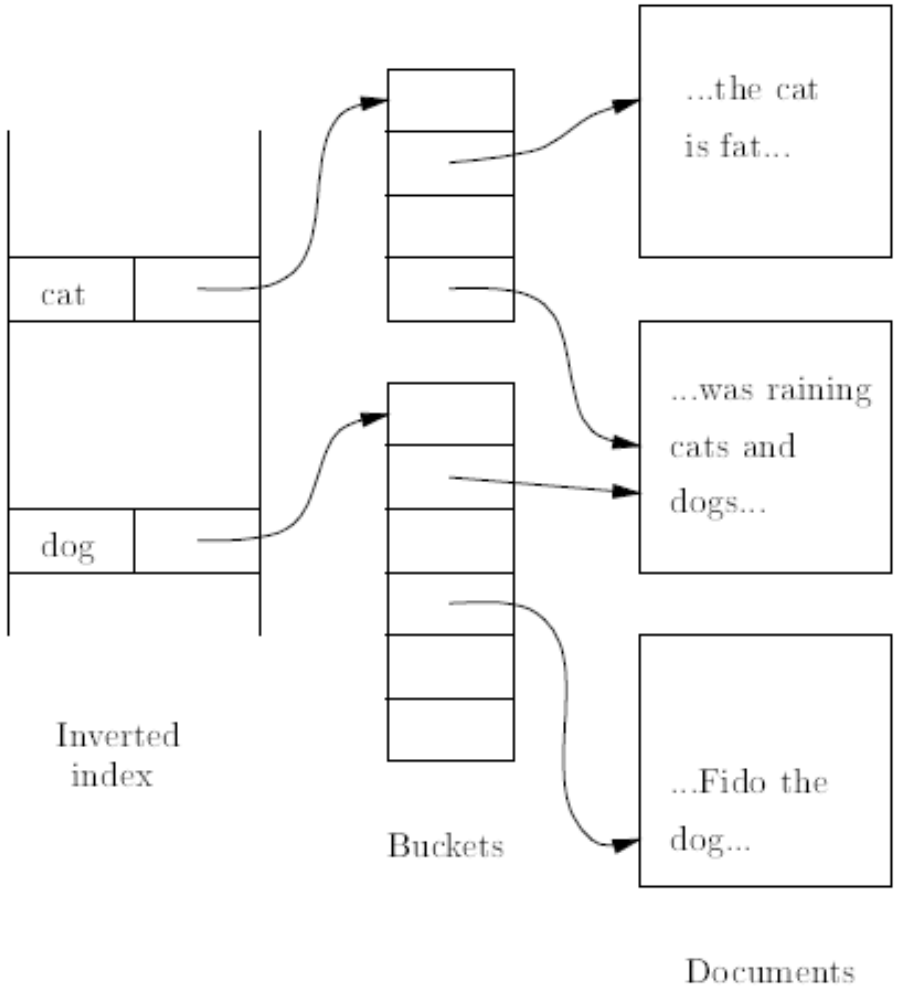
Indice Secundario



Indice invertido

- En lugar de un archivo de datos, existe una colección de documentos
- Las entradas en el índices se componen de: (1) palabras que constituyen una clave de búsqueda, (2) puntero a un bucket.
- Un bucket contiene punteros a los bloques de disco en donde se pueden encontrar documentos que contiene la clave de búsqueda.

Indice Invertido y Recuperación de Documentos



Stemming y Stop words

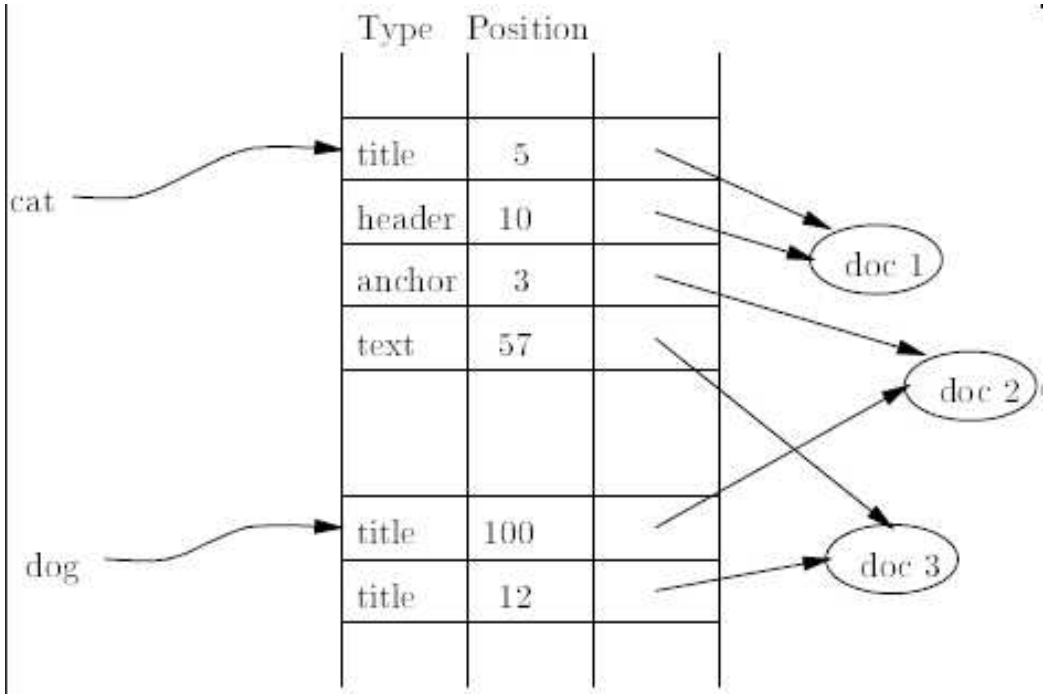
- Constituyen técnicas básicas para mejorar la efectividad del índice.
- Stemming: la idea es reducir palabras derivadas y considerar sólo su raíz. Por ejemplo: **pesca**dor, **pesca**do, **pesca**ndo, **pesca**.
- Stop words: consiste en eliminar artículos, pronombres, proposiciones, etc. ya que aparecen en todos los documentos. La eliminación de stop words reduce considerablemente el tamaño del índice.

La incorporación de más datos dentro de un bucket permite obtener documentos sin tener que examinarlos en detalle.
Por ejemplo, se pueden recuperar todos los documentos sobre perros que realizan una mención a gatos.
Puede que la respuesta no sea la más adecuada pero sin duda constituye un gran acercamiento a la respuesta si consideramos la búsqueda de documentos donde:

- La palabra perro aparece en el título.
- Se menciona gato en un enlace.

Se puede responder a esta consulta intersectando los punteros.

Indice Invertido y Recuperación de Documentos



Indice basado en árbol B

Arbol B

Corresponde a una generalización de un índice multinivel frecuentemente utilizado en DBMS comerciales. La variante más utilizada de árbol B es el árbol B+. Básicamente un árbol B:

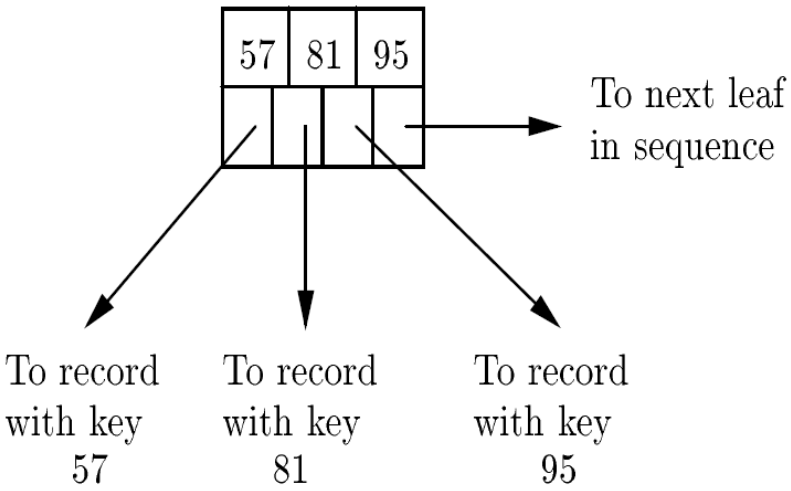
- considera automáticamente una cantidad de niveles en el índice.
- administra los bloques de tal manera que cada bloque es utilizado entre un 50% y un 100%. De este modo no es necesario considerar bloques de overflow.

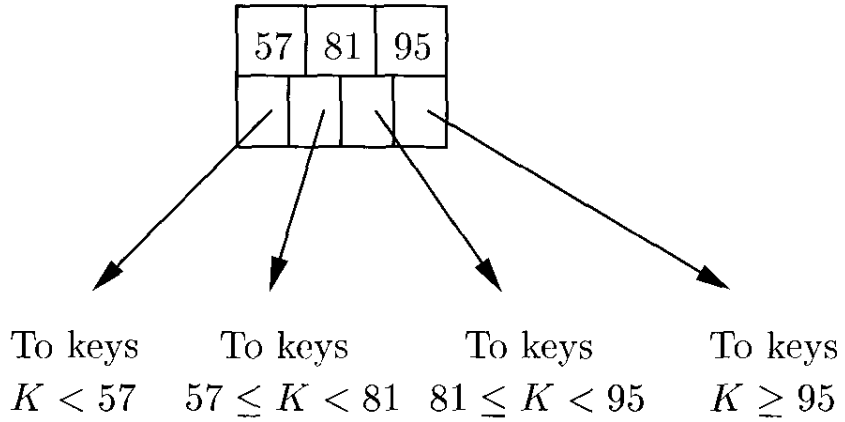
Arbol B: estructura

Un árbol B está compuesto de 3 niveles: raíz, nivel intermedio, hojas.
Además, un árbol B es balanceado, de este modo las rutas entre la raiz y los nodos hoja tienen el mismo largo.
Cada nodo del árbol está compuesto de n claves de búsqueda y $n+1$ punteros.
El valor de n debe ser tal que un nodo pueda ser almacenado en forma completa dentro de un bloque de disco.

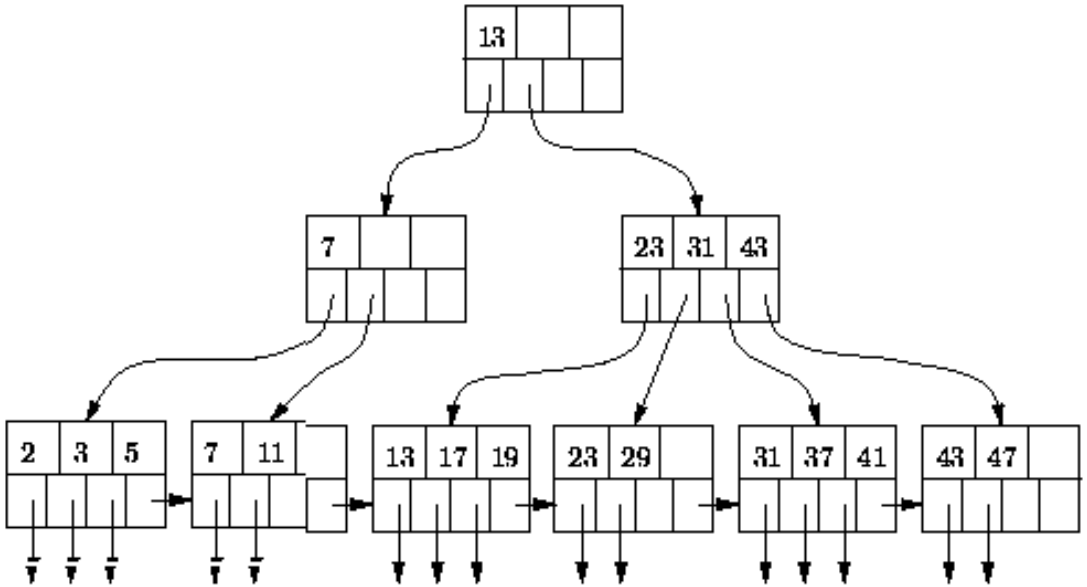
Arbol B: estructura

- En la raíz hay al menos 2 punteros los cuales apuntan a bloques del nivel inferior.
- En un nodo hoja el último puntero apunta al siguiente nodo hoja ubicado a su derecha. Al menos $\frac{n+1}{2}$ son utilizados para apuntar a bloques de datos.
- En un nodo interior si existen en uso j punteros existirán $j-1$ claves $K_1, K_2, ..., K_{j-1}$. Los nodos de un árbol B desde izquierda a derecha se encuentran en orden no decreciente.





Indice basado en árbol B



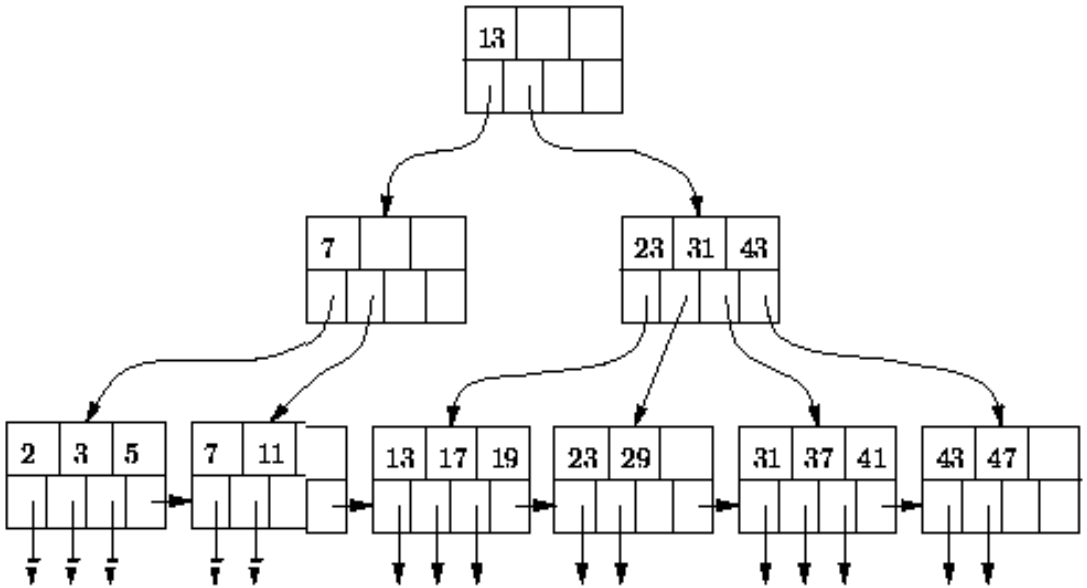
Indice basado en árbol B

Arbol B

- Clave de búsqueda es clave primaria → índice denso. No necesariamente el archivo de datos se encuentra ordenado basado en la clave primaria.
- Archivo de datos ordenado basado en clave primaria → índice no denso.
- Clave de búsqueda no es clave primaria y se usa para ordenar archivo de datos → en un nodo hoja el puntero apunta al bloque donde se encuentra la primera ocurrencia.

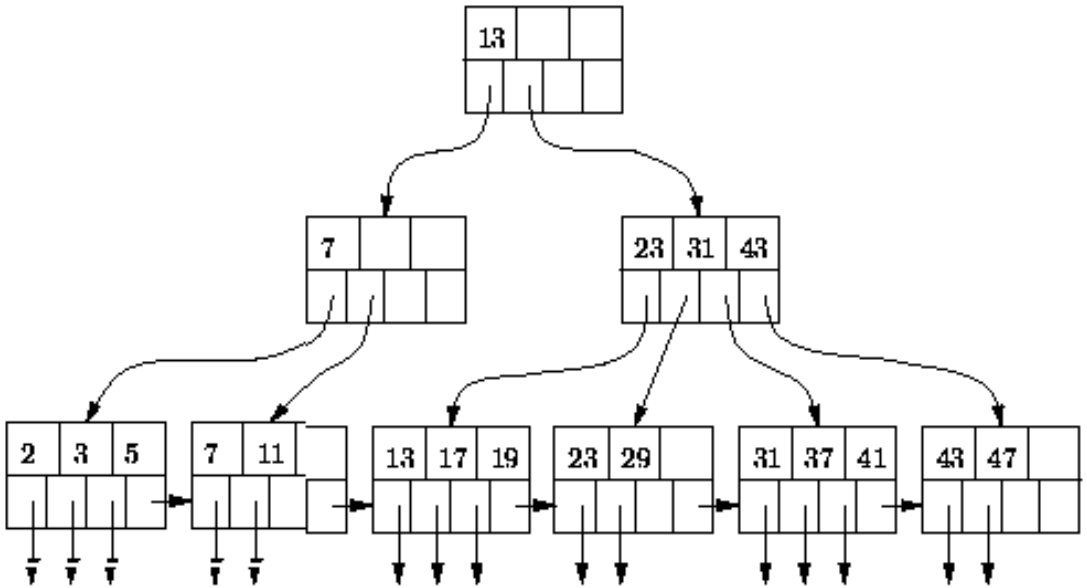
Arbol B

- al buscar registros con la clave 13, el punto de entrada es el subárbol izquierdo.
- qué sucede si se desea recuperar registros $24 \leq K_i \leq 36$?



Arbol B: range queries

- se utilizan para buscar registros cuya clave pertenece al rango $[a, b]$
- qué sucede si b es ∞?
- qué sucede si a es $-\infty$?



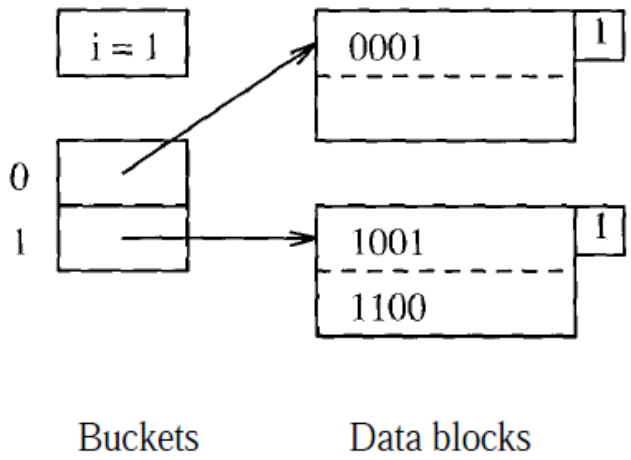
Indice basado en árbol B

Ejercicio

Se tiene un archivo de 200000 registros cada uno de 250 bytes. Justifique cuál es la mejor elección, para la recuperación de registros, si se desea construir un índice sobre la clave del archivo (9 bytes)?
Considere: índice primario, índice basado en árbol B (ptr de 3 bytes). Disco: 3600 rpm (delay), 18ms (seek) y 1229 bytes/ms (transferencia), tamaño del bloque: 1KB.

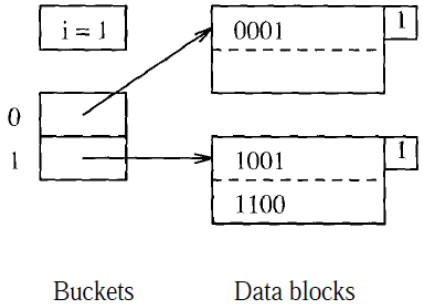
Hashing extendido

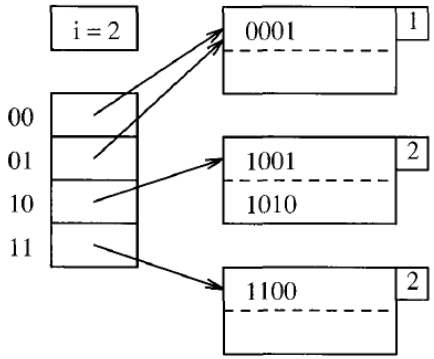
- una función de hash h calcula para cada clave una secuencia de k bits.
- Sin embargo, sólo se usan d bits
- La estructura de acceso es un arreglo de 2^d direcciones de buckets.
- d es la profundidad global del directorio.
- Se suponen buckets de tamaño fijo.
- En caso de overflow es necesario doblar el tamaño del directorio.
- La estructura de acceso se construye considerando el bit más significativo.

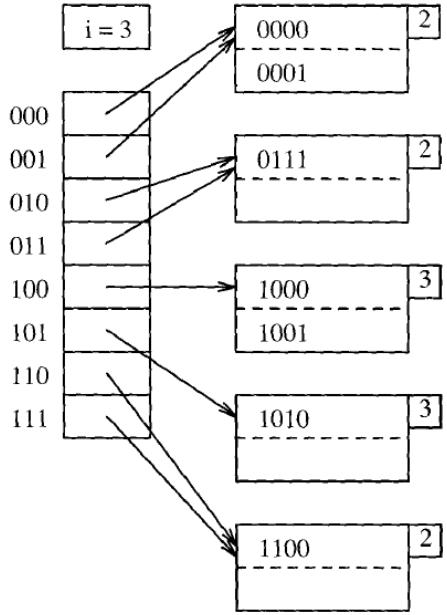


Hashing extendido: inserción

- La inserción de 1010 obliga a doblar la cantidad de entradas en el directorio (ver imagen izquierda).
- La inserción de 0000 y 0111 implica agregar un bloque pero no agregar más entradas en el directorio.
- La inserción de 1000 implica $d = 3$ (ver imagen derecha).

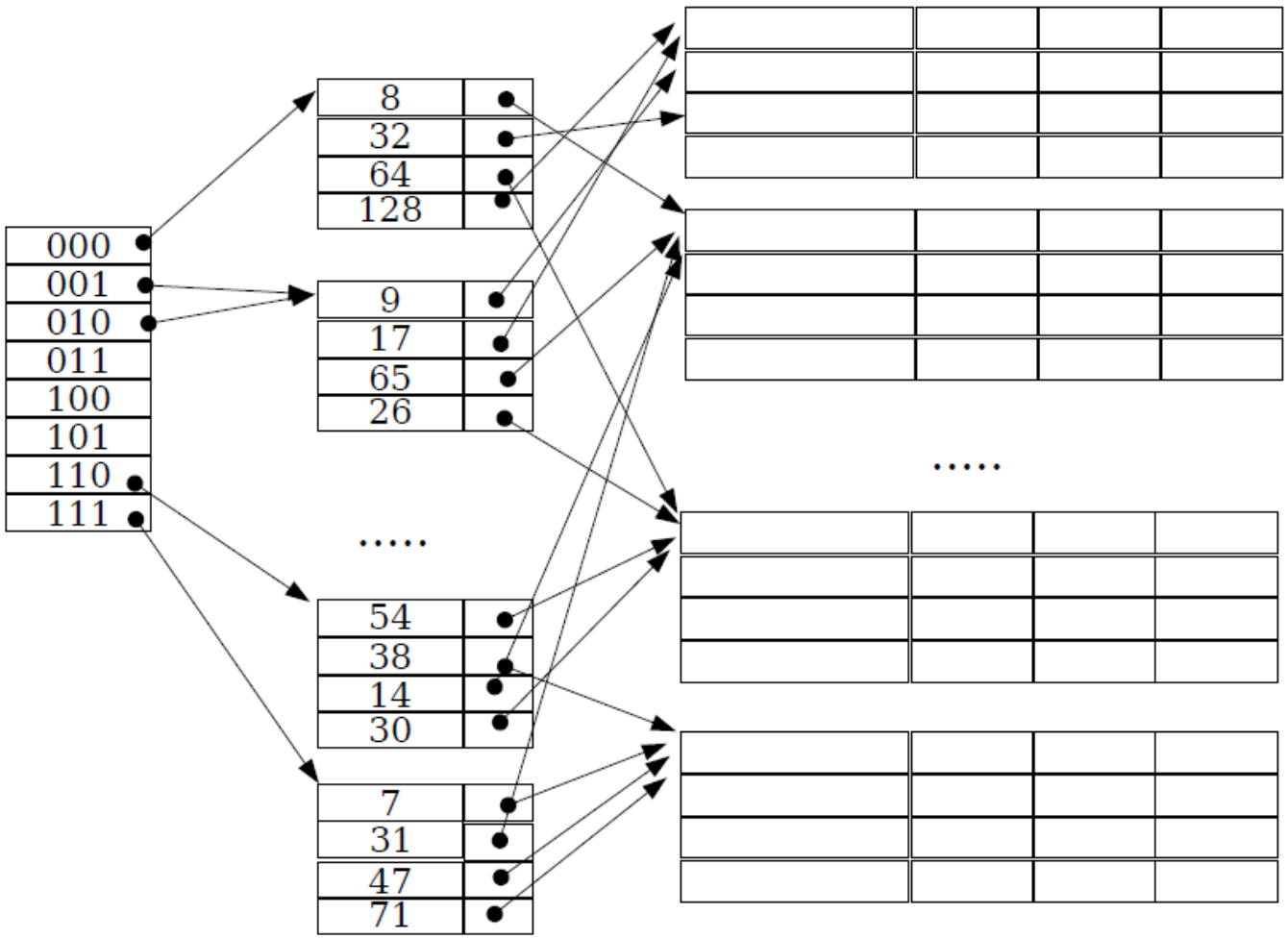






Hashing extendido

para buscar se utiliza k mod 2^d



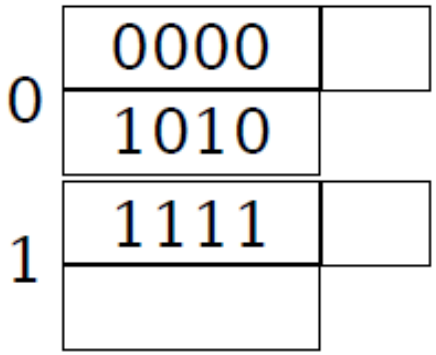
Hashing lineal

- No es necesario el uso de un directorio.
- Comienza con 2^d buckets. d es la cantidad de bits utilizados para determinar la pertenencia a un bucket. La cantidad de buckets crece gradualmente.
- Incremento de buckets puede ser gatillado por la existencia de **overflow** o cuando se ha alcanzado un **factor de carga**.
- Factor de carga (fc):
  - corresponde al número de registros almacenados dividido por el número de posibles ubicaciones de almacenamiento.
  - la cantidad inicial de buckets n es una potencia de 2. Sin embargo, su crecimiento no siempre es en base a una potencia de 2.
  - las ubicaciones de almacenamiento: $s = \#buckets \times \#regs/bucket$
  - la cantidad inicial de registros r es 0 y se incrementa cuando se inserta un registro.
  - $fc = \frac{r}{s} = \frac{r}{(n \times \#regs/bucket)}$
- Si $fc > 90\%$ el espacio de almacenamiento se debe incrementar.

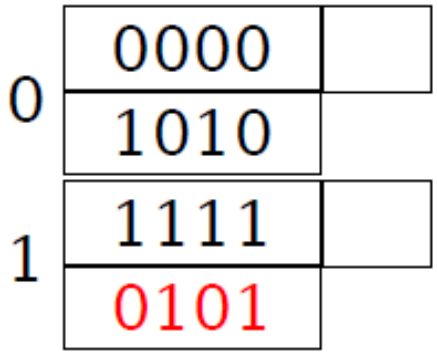
Hashing lineal: algoritmo de inserción

- se inserta un registro con clave K sólo cuando se ha calculado el valor de la función de transformación H.
- se toman los últimos d dígitos de H.
- buscar el bucket m donde K debe ser almacenada.
- si $(m < n) \rightarrow$ existe un bucket para almacenar K, si el bucket no tiene espacio usar bucket de overflow.
- si $(m \geq n) \rightarrow$ almacenar K en el bucket $m - 2^{d-1}$.
- luego de cada inserción chequear si $fc < umbral$.
- si $(fc > umbral) \rightarrow$ incrementar el espacio de almacenamiento (split) considerando:
  - agregar un nuevo bucket (esto puede incrementar d).
  - redistribuir los registros entre el nuevo bucket $n = 1b_2...b_d$ y el bucket $n = 0b_2...b_d$.

Ejemplo: considere que H genera un valor compuesto de 4 dígitos binarios. $n = 2$,$d = 1$,$fb = 2$,$r = 3$. Inicialmemte se insertan 0000,1010 y 1111.



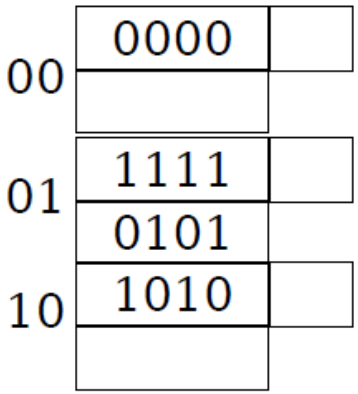
Insertar 0101



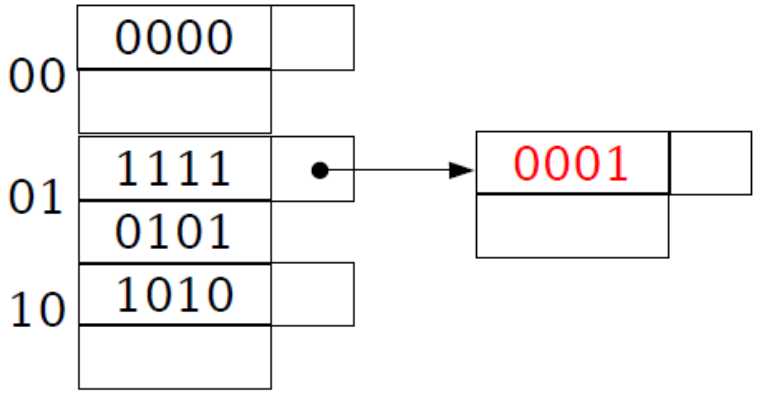
Luego de la inserción $fc = \frac{4}{4} = 100\% \rightarrow split$!

Indice basado en Hashing

Se agrega el bucket 10 y las claves se redistribuyen. $d = 2, n = 3, r = 4$

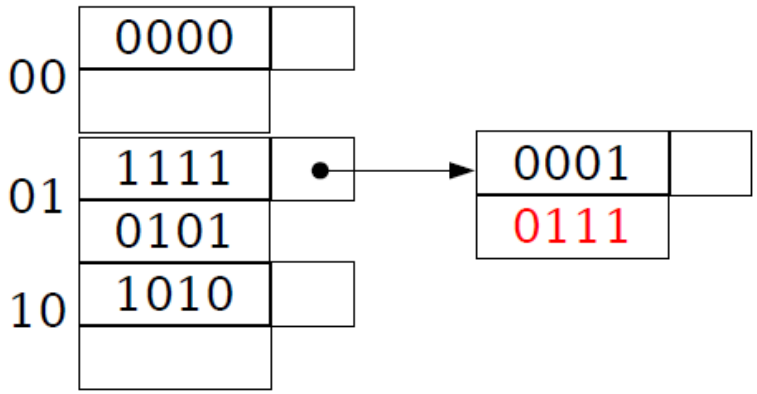


Insertar 0001. $01 < 10$, pero el bucket no tiene espacio $\rightarrow$ bucket de overflow. $d = 2, n = 3, r = 5$



Indice basado en Hashing

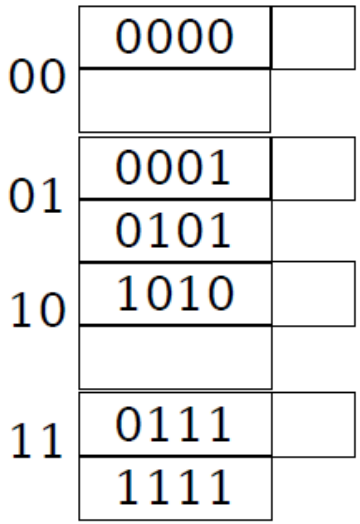
Insertar 0111, pero $11 < 10$ entonces se inserta 0111 en el bucket $m - 2^{d-1} = 11 - 10 = 01$. $d = 2, n = 3, r = 6$



luego de la inserción $fc = \frac{6}{6} = 100\% \rightarrow split!$

Indice basado en Hashing

Se agrega el nuevo bucket 11 y se reparten los registros con bits 01 y 11.
$d = 2, n = 4, r = 6$



Indices: resumen

- **Archivos secuenciales**. organización simple, generalmente ordena los datos en base a la clave primaria de los registros y construye un índice para acceder a ellos.
- **Indices denso - no denso**. un índice denso contiene una entrada por cada registro del archivo. Un índice no denso contiene una entrada por cada bloque del archivo.
- **Indice multinivel**. corresponde a la construcción de un índice no denso sobre un índice. Resulta útil cuando el índice no puede ser almacenado en memoria principal.
- **Indice secundario**. es un índice denso basado en un atributo que no es clave. El índice se ordenada en base al atributo de indexación.
- **Indice invertido**. generalmente utilizado en la recuperación de documentos. Representa la relación entre documentos y las palabras que contienen. Sus entradas apuntan a un bucket donde a su vez existe una lista de punteros hacia los archivos donde existe una ocurrencia de una palabra.
- **Arboles B**. por definición es un índice multinivel que permite realizar consultas por rango. Es el más utilizado en administradores de bases de datos.
- **Indices basados en hashing**. no permiten consultas por rango.

Indices: Criterios para la selección de índices

El diseño físico de una BD tienen como actividad la elección de las estructuras de almacenamiento y formas de acceso a los datos, con el objetivo de obtener un buen rendimiento.

Aspectos a considerar

- Tiempo de respuesta
- Espacio en disco: dimensionar el espacio requerido.
- Rendimiento del sistema

Tareas

- Analizar las consultas sobre los datos: archivos involucrados, atributos sobre los cuales existen condiciones de selección, atributos sobre los cuales hay condiciones de selección que involucran a registros de otros archivos.
- Análisis de frecuencia esperada de la invocación de la consulta
- Análisis de posibles restricciones de tiempo de las consultas.
- Análisis de la frecuencia esperada de operaciones de actualización

Método para realizar el diseño físico

- Una opción común es mantener los registros desordenados y crear muchos índices secundarios sobre atributos que se utilizan para operaciones de búsqueda y join.
- Otra alternativa es:
  - escoger un atributo usado frecuentemente y definirlo como campo de ordenamiento para crear un índice primario o agrupado (según corresponda) sobre él.
  - por cada atributo, que no es el campo de ordenamiento, y que es frecuentemente utilizado en operaciones de búsqueda y join, especificar un índice secundario.
  - si las actualizaciones son muy frecuentes se debe reducir el número de índices.
  - si un atributo es usado frecuentemente en operaciones de búsqueda y join pero no para recuperar registros en orden, un archivo tipo hashing puede ser utilizado, de lo contrario se puede usar árbol b+.