Teórico 11

Indexación y Hashing

Conceptos básicos

- Los mecanismos de indexación son usados para acceder rápidamente a los datos deseados.
  - E.j., autor en un catalogo de libros
- Clave de búsqueda: es un atributo o conjuntos de atributos usados para buscar registros en un archivo.
- Un archivo de índices esta compuesto de registros (llamados entrada de índices) de la forma:

Clave de búsqueda	puntero

- Los archivo de índices son generalmente mucho mas pequeños que el archivo original.
- Hay dos clases básicas de índices:
  - Índices ordenados: Las claves de búsquedas están almacenadas ordenadas.
  - Índices de Hash: Las claves de búsquedas están distribuidas uniformemente a través de “cajones” utilizando una “función de hash”.

Evaluación de Métricas para Índices

Las técnicas de indexación son evaluadas en base a:

- Tipos de accesos soportados eficientemente. Por ej.:
  - Registros con un valor especifico para un atributo.
  - Registros con un valor en un rango determinado.
- Tiempo de acceso.
- Tiempo de Inserción.
- Tiempo de borrado.
- Espacio adicional requerido.

Índices Ordenados

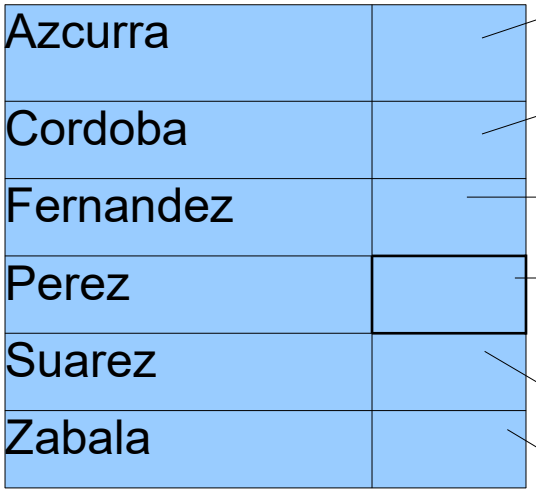
- En un **índice ordenado,** las entradas de índices son almacenadas ordenadas por el valor de clave de búsqueda. Ej., catalogo de Autor en una librería.
- **Índices Primarios:** en un archivo ordenado secuencialmente, es el índice cuya clave de búsqueda especifica el orden secuencial del archivo.
  - La clave de búsqueda de un índice primario es usualmente, pero no necesariamente, la clave primaria.
- **Índices secundarios**: es un índice cuya clave de búsqueda especifica un orden diferente del orden secuencial del archivo.

Índices Primarios

Archivos de índices densos

Índices Densos — aparece un registro índice para cada valor de la clave de búsqueda en el archivo.

En la siguiente figura se muestra un índice denso para la tabla (archivo) cuenta:



Edad	Apellido	Nombre	Saldo
34	Azcurra	Jorge	4545
23	Cordoba	Luis	432
54	Cordoba	Carlos	878
34	Fernandez	Beatriz	-457
45	Perez	Juan	96
54	Perez	Jorge	0
65	Perez	Evelio	0
86	Suarez	Hernan	21
12	Zabala	Jose	45

Archivos de Índices Dispersos

- Índices dispersos: contienen registros índices sólo para algunos valores de la clave de búsqueda.
  - Aplicable cuando los registros del archivo están ordenados secuencialmente en la clave de búsqueda.
- Para localizar un registro con valor de clave de búsqueda K:
  - Buscar el registro de índice con el valor mas grande que sea menor o igual que el valor que se esta buscando.
  - Buscar en el archivo secuencialmente comenzando en el registro que apunta la entrada de índice seleccionada.
- Menos espacio y menos overhead de mantenimiento para inserciones y borrados.
- Generalmente mas lento que un índice denso para localizar registros.
- Buen Rendimiento: los índices dispersos con una entrada por cada bloque en el archivo, esta entrada se corresponde con el menor valor de clave en el bloque.

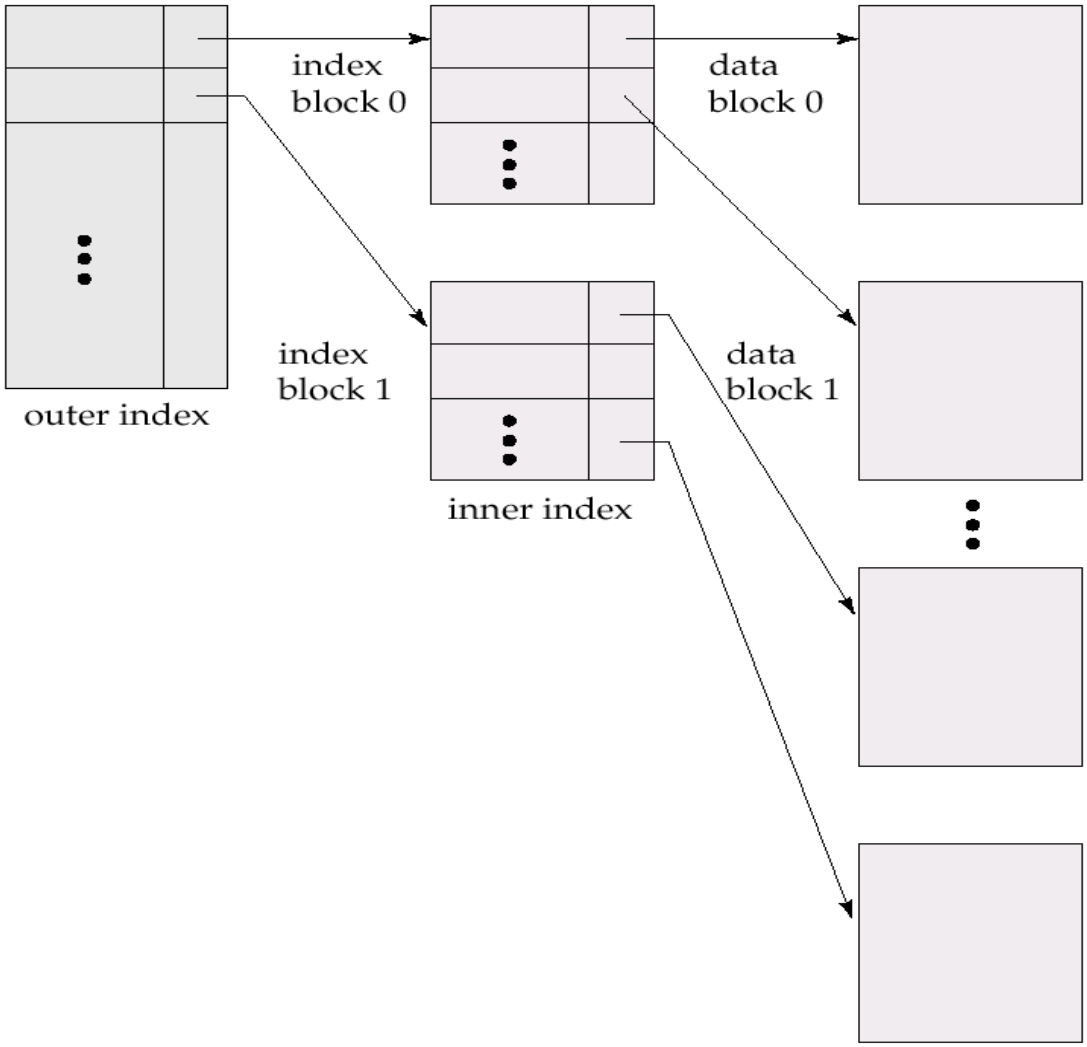
Ejemplo de Archivo de Índices Dispersos

Edad	Apellido	Nombre	Saldo
34	Azcurra	Jorge	45
23	Cordoba	Luis	400
54	Cordoba	Carlos	878
35	Fernandez	Beatriz	-457
45	Perez	Juan	400
57	Perez	Jorge	0
65	Perez	Evelio	0
86	Suarez	Hernan	400
12	Zabala	Jose	45

Índices Multinivel

- Si el índice primario no cabe en memoria, el acceso llega a ser muy costoso.
- Para reducir el número de accesos a disco, se trata el índice primario con un archivo secuencial y se construye un índice disperso sobre el índice primario.
  - Índice externo –un índice disperso del índice primario
  - Índice interno – el archivo de índice primario
- Si el índice externo es tan grande que no cabe en memoria, otro nivel de índices puede ser creado y así sucesivamente.
- Los índices en todos los niveles deben ser actualizados en la inserción o borrado de un registro en el archivo.

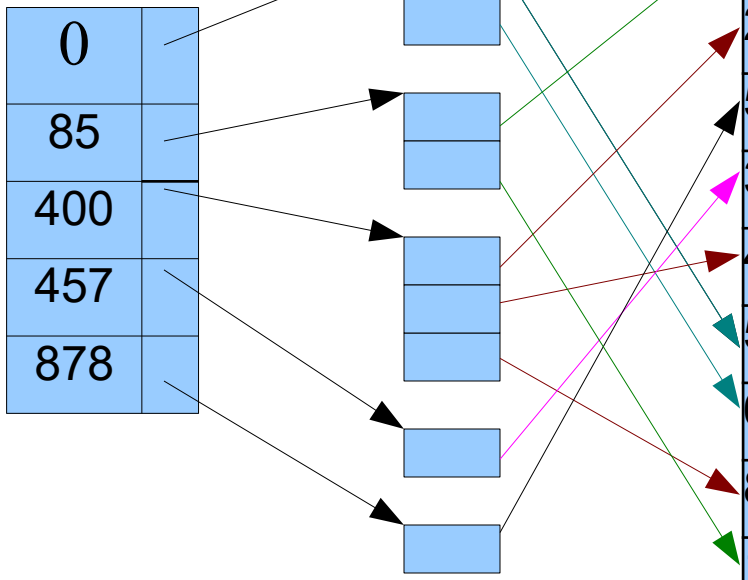
Índice Multinivel (Cont.)



Índices Secundarios

- Frecuentemente, uno quiere encontrar registros en un archivo(tabla) cuyos valores en un cierto campo(el cual no es la clave de búsqueda del índice primario) satisface alguna condición.
  - Ejemplo 1: En una tabla cuentas bancarias almacenadas secuencialmente por número de cuenta, nosotros podemos querer encontrar todas las cuentas de una determinada sucursal o todas las cuentas con un determinado saldo.
  - Ejemplo 2: En una tabla personas que tiene clave primaria DNI, podemos querer hacer búsquedas por Apellido
- Nosotros podemos tener un índice secundario con un registro índice por cada valor de clave, Los registros índices apuntan a cajones que contienen punteros a todos los registros con esa clave.

Índice Secundario Denso en el Campo Saldo de la Tabla Titular



Cuenta	Apellido	Nombre	Saldo
34	Azcurra	Jorge	85
23	Cordoba	Luis	400
54	Cordoba	Carlos	878
35	Fernandez	Beatriz	457
45	Perez	Juan	400
57	Perez	Jorge	0
65	Perez	Evelio	0
86	Suarez	Hernan	400
12	Zabala	Jose	85

Índices Primarios y Secundarios

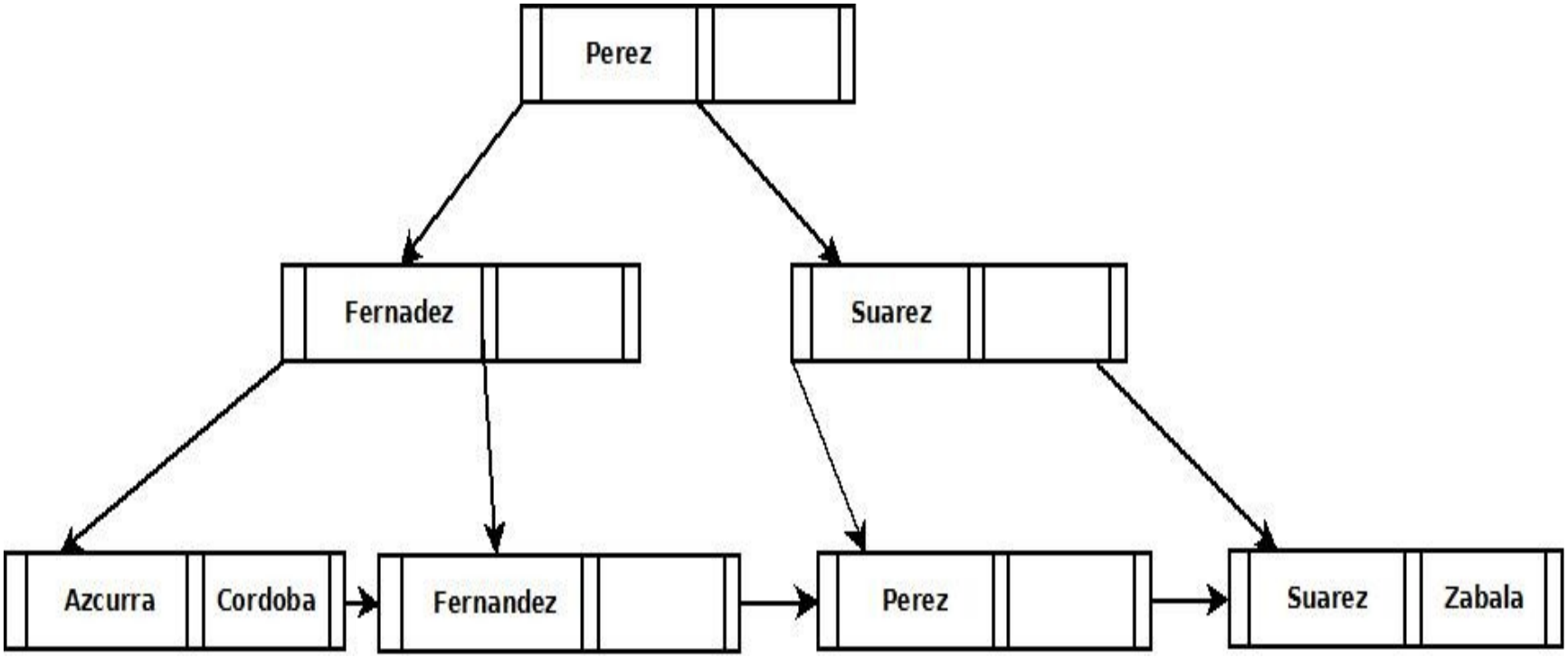
- Los índices secundarios tienen que ser denso, en el sentido que tiene que haber un puntero en los cajones a cada registro del archivo.
- Los índices ofrecen beneficios sustanciales al buscar registros.
- Cuando un archivo se modifica, cada índice en el archivo debe actualizarse. Actualizar los índices impone un overhead en la modificación de la base de datos.
- Hacer un scaneo secuencial usando índice primario es eficiente, pero un scaneo secuencial usando un índice secundario es ineficiente.
  - cada acceso a un registro puede tener que recuperar un nuevo bloque del disco.

Archivos de Índice B$^+$-Tree

Los índices con árboles B$^+$ son una alternativa a los archivo de índices secuenciales.

- Desventajas de los archivo indexados secuencialmente: la performance se degrada cuando el archivo crece. La reorganización periódica del archivo completo es necesaria, pero esto no es deseable.
- Ventaja de archivos de índice de árbol B+: automáticamente se reorganiza con cambios pequeños y locales, ante las inserciones y eliminaciones. La reorganización de archivo entero no se exige para mantener la performance de los archivos de índices con árboles B+-.
- Desventajas de árboles B+: overhead en la inserción y eliminación y también tenemos un overhead de espacio.
- Las ventajas de los árboles B+ valen más que las desventajas, los árboles B+ son muy utilizados.

Ejemplo de Árbol B+

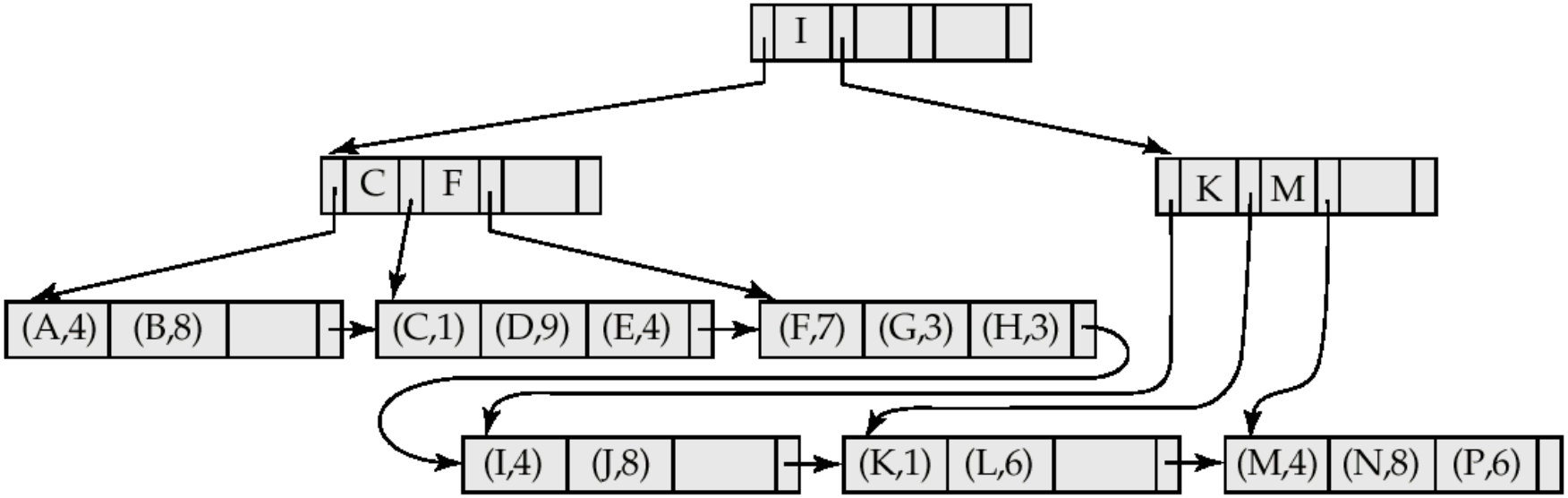


Árbol B+ para el archivo Titular (*n* = 3)

Organización de Archivos con Árboles B+

- La degradación de los archivos de índices se resuelve con la utilización de los índices con B+. La degradación de los archivos de datos se resuelve con el uso de árboles B+ para la organización de los archivos.
- Los nodos hojas en un organización de archivos de árbol B+ almacenan registros en lugar de punteros.
- Dado que los registros son más grandes que los punteros, el número máximo de registros que pueden guardarse en un nodo hoja es menor que el número de punteros en un nodo no hoja.
- Los nodos hojas deben estar medios llenos.
- La inserción y borrado se manejan de la misma forma que las inserciones y borrados de las entradas en un índice con Árbol B+.

Organización de Archivos con Árboles B+ (Cont.)



Ejemplo de una organización de archivos Árboles B+

- Buena utilización del espacio dado que los registros usan más espacio que los punteros.
- Para mejorar utilización del espacio, hay que involucrar mas nodos hermanos en la redistribución durante divisiones y fusiones

Archivos de Índices con Árboles B

- Similares a los árboles B+, pero los árboles B permiten que las claves de búsquedas aparezcan sólo una vez, de esta forma se elimina la redundancia en el almacenamiento de valores claves.
- Las claves de búsqueda en nodos no hojas no vuelven a aparecer en otra parte del árbol; un campo puntero adicional para cada clave de búsqueda debe ser incluido en los nodos no hojas

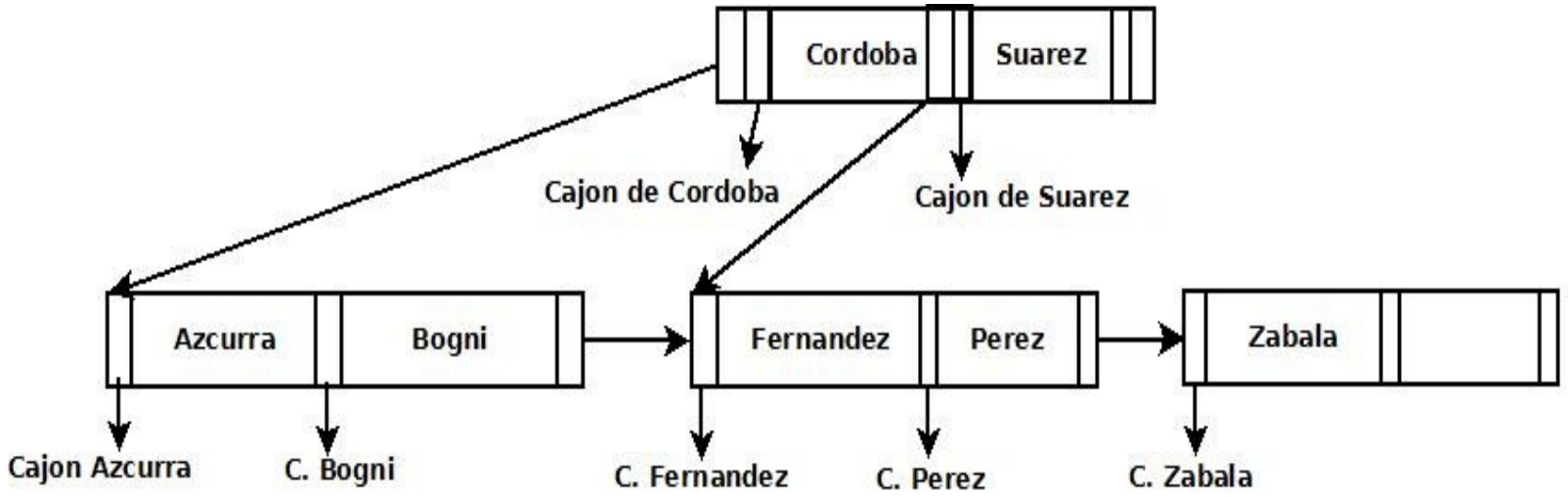
a) Nodo hoja

(a)

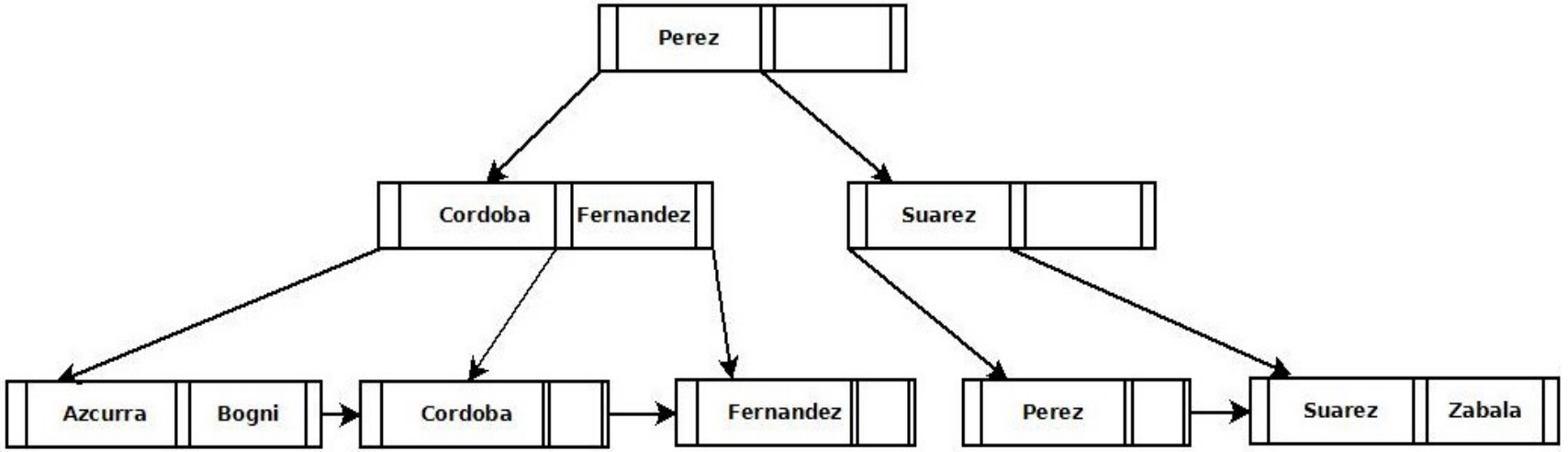
(b)

b) En los nodos no hojas – los punteros Bi son punteros a un registro de un archivo o a un cajón.

Ejemplo de indice de Arbol B y B++



Árbol B (arriba) y árbol B+ (abajo) con los mismos datos



Archivos de Índices de Árbol B (Cont.)

- Ventajas de índices con árbol B:
  - Pueden usar menos nodos que un árbol B+.
  - Algunas veces se pueden encontrar el valor de clave antes de llegar a un nodo hoja.
- Desventajas de índices con árboles B:
  - Sólo unos pocos valores buscados se encuentran rápido.
  - Las inserciones y borrados son mas complicados que en los árboles B+
  - La Implementación es más difícil que los árboles B+.
- Los implementadores de base de datos prefieren la sencillez de los árboles B+.

Hashing Estático

- Un cajón (bucket en ingles) es una unidad de almacenamiento que contiene uno o más registros (un cajón es típicamente un bloque de disco).
- En una organización de archivos por hashing se obtiene el cajón de un registro directamente de su valor de clave de búsqueda utilizando la función de hash.
- La función de hash h es una función del conjunto de valores de todas las claves de búsqueda Hash función K al conjunto de todos los cajones direccionables B.
- La función de hash es utilizada para localizar registros para su acceso, inserción y borrado.
- Registros con diferentes valores de claves de búsqueda pueden ser mapeados al mismo cajón, de esta forma una vez accedido el cajón se debe hacer una búsqueda secuencial para encontrar el registro.

Ejemplo de Organización de Archivos por Hash

- La Organización de archivos por Hash del archivo Titular, utilizando Apellido como clave

Cajon 0

Cajon 1

Cajon 2

Cajon 3

34	Azcurra	Jorge	85
12	Zabala	Jose	85

Cajon4

86	Suarez	Hernan	400

Cajon 5

45	Perez	Juan	400
54	Perez	Jorge	0
65	Perez	Evelio	0

Cajon 6

Cajon 7

34	Fernandez	Beatriz	457

Cajon 8

23	Cordoba	Luis	400
54	Cordoba	Carlos	878

Cajon 9

Funciones de Hash

- La peor función mapea todos los valores de claves de búsqueda al mismo cajón.

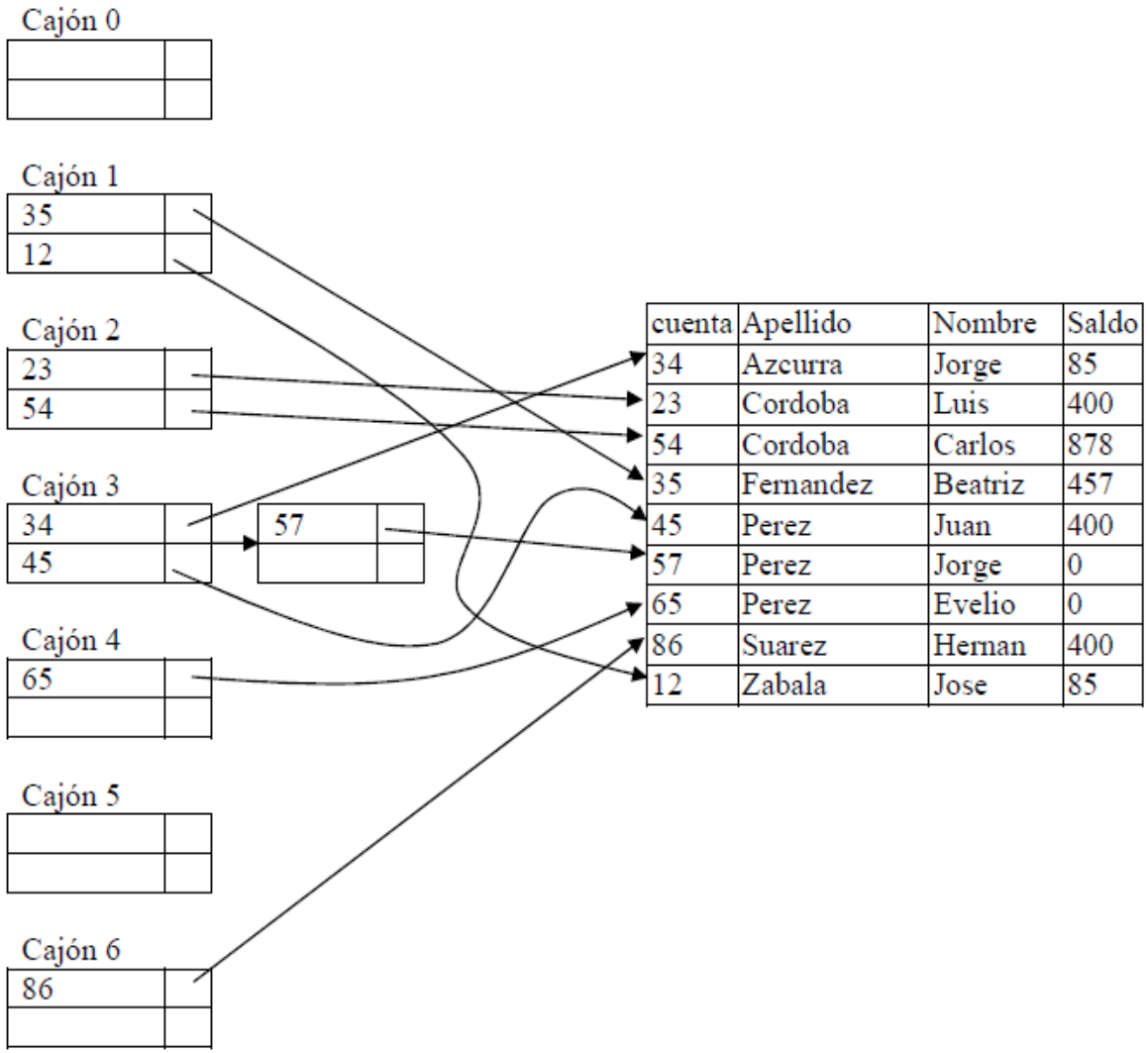
- Una función de hash ideal es uniforme, es decir, cada cajón tiene asignado el mismo número de valores de claves de búsqueda del conjunto de todos los posibles valores.

- Una función de hash ideal es random, así cada cajón tendrá la misma cantidad de registros asignados, la función deberá parecer aleatoria, no deberá seguir ningún orden externo visible (lexicográfico, etc).

Índices de Hash

- El hashing se puede usar no sólo para la organización de archivos, sino también para la creación de la estructura de índices.
- Un índice hash organiza claves de búsqueda, con sus punteros a registros asociados, dentro de una estructura de archivo de hash.
- Estrictamente hablando, los índices de hash siempre son índices secundarios.
  - Si el propio archivo es organizado con hash, un índice de hash primario separado usando la misma clave de búsqueda es innecesario.

Ejemplo de un Índice de Hash sobre atributo cuenta de la tabla titular



cuenta	Apellido	Nombre	Saldo
34	Azcurra	Jorge	85
23	Cordoba	Luis	400
54	Cordoba	Carlos	878
35	Fernandez	Beatriz	457
45	Perez	Juan	400
57	Perez	Jorge	0
65	Perez	Evelio	0
86	Suarez	Hernan	400
12	Zabala	Jose	85

Deficiencias del Hashing Estático

- En el hashing estático, la función h mapea los valores de clave de búsqueda a un conjunto fijo B de cajones.
  - Las bases de datos crecen con el tiempo. Si el número inicial de cajones es demasiado chico, la performance se degradará porque abra demasiados cajones de desbordamiento.
  - Si el tamaño del archivo va ser grande en un futuro, se pueden tener un gran numero de cajones, pero así se estará desperdiciando espacio al principio.
  - Si la base de datos se achica, nuevamente se estará gastando espacio.
  - Una opción es reorganizar periódicamente el archivo con una nueva función de has, pero esto es muy caro.
- Este problema puede ser evitado utilizando técnicas que permitan que el numero de cajones sean modificados dinámicamente.

Hashing Dinámico

- Bueno para base de datos que crecen y se achican con el tiempo
- Permiten modificar la función de hash dinámicamente
- Hashing Extensible– una forma de hashing dinámico.
  - La función de Hash genera valores sobre un rango relativamente amplio — típicamente b-bit, con b = 32.
  - Siempre se usa sólo un prefijo de la función de Hash para indexar dentro de una tabla de cajones direccionables.
  - La longitud del prefijo es de i bits, 0 <= i <= 32.
  - El tamaño de la tabla de cajones direccionables es de 2i. Inicialmente i = 0
  - El valor de i crece y se achica al mismo tiempo que crece y se achica la base de datos.
  - Varias entradas en la tabla de cajones pueden apuntar al mismo Cajón.
  - Así, el número actual de cajones es menor que 2i
    - La cantidad de cajones también cambian dinámicamente ya que se producen fusiones y divisiones de cajones

Definición de Índices en SQL

- Creación de un Índice

  CREATE INDEX <nombre-índice> on <nonbre-tabla>
  <lista-atributos>)

  Ej.: CREATE INDEX titular-indice ON titular(apellido)

- Usar create unique index para especificar y forzar la condición de que la clave de búsqueda es una clave candidata.

- Para borrar un índice:

  DROP INDEX <nombre-índice>

Accesos Multiple-Claves

- Usar índices múltiples para determinados tipos de queries.
- Ejemplo:

  select cuenta

  from Titular

  where Apellido = “Perez” and saldo = 500

- Posibles estrategias para procesar este query índices en atributos simples:

  1. Usar el índice sobre Apellido para buscar cuentas con Apellido = “Perez” ; testear saldo = $500.
  2. Usar el índices sobre saldo para buscar cuentas con balances de $500; testear Apellido = “Perez”.
  3. Usar el índice Apellido para buscar los punteros a todos los registros pertenecientes a Perez. De la misma forma usar el índice sobre saldo. Tomar la intersección de ambos conjuntos punteros.

Índices sobre Múltiples Atributos

El orden de los valores de la clave de búsqueda es el orden lexicógrafo

Supongamos tener un índice combinado sobre la clave de búsqueda (Apellido*, Saldo*).

- Con la cláusula

  where *apellido* = “Perez” **and** saldo = *500*
  El índice sobre la clave de búsqueda combinada buscará sólo los registros que cumplan ambas condiciones. Utilizando índices separados es menos eficiente — Podemos encontrar muchos registros que cumplen sólo una de las condiciones.

- Puede manejar eficientemente

  where *apellido* = “Perez” **and** saldo < 500

- Pero no puede manejar eficientemente
  where *Apellido* < “Perez” **and** saldo = 500
  Puede encontrar muchos registros que cumplen la primera pero no la segunda condición.

Bases de Datos Avanzadas

- Bases de Datos NoSQL
- Datawarehousing.
- Bases de Datos Geograficas.
- OnLine Analitical Processing (OLAP).
- Data Mining.
- Bases de Datos Temporales.
- Bases de Datos Espaciales.
- Big Data.