Bases de Datos 2

Índices

Ingeniería en Informática

Facultad Politécnica – UNA

Prof. Ing. Joaquín Lima

Conceptos Básicos

- Los índices son estructuras de una BD utilizados para acelerar el acceso a los datos deseados.
- **Clave de Búsqueda**
  - Atributo (columna) o conjunto de atributos en base al cual se construye el índice y a través de cuyos valores se obtienen los datos.
- **Archivo Índice**
  - Archivo que contiene **registros índice** formados por un campo que almacena un valor de la clave de búsqueda y un puntero o conjunto de punteros a las fila/s de la tabla

- Existen dos tipos de índices
  - Índices Ordenados:
    - Los **registros índice** son almacenados según el orden definido por la clave de búsqueda
  - Índices Asociativos
    - Los **registros índice** son almacenados en un cajones (buckets) partir del uso de una función de hash o de asociación

Evaluación de técnicas de indexación

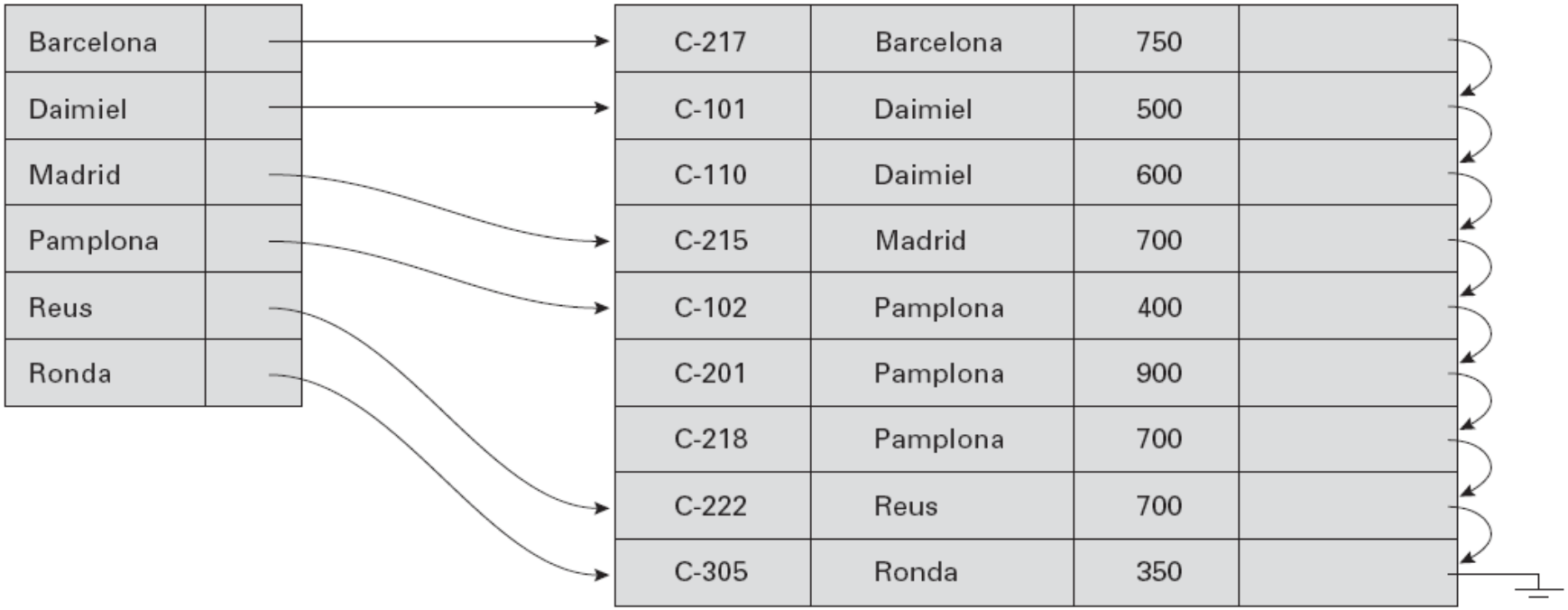
- Existen varias técnicas de indexación, cada una de ellas es más apropiada para un caso en particular.
- Por ello cada técnica debe ser valorada según los siguientes criterios.
  - Tipo de acceso.
    - Búsqueda de un valor concreto
    - Búsqueda de un rango de valores
  - Tiempo de Acceso
  - Tiempo de Inserción
  - Tiempo de Borrado
  - Espacio adicional requerido

Índices Ordenados

- Los registros índices están ordenados de acuerdo al valor del la clave de búsqueda.
- Existen dos tipos de índices ordenados
  - **Índices Primarios**
    - Son aquellos índices cuya clave de búsqueda especifica el orden secuencial del archivo indexado.
      - **Archivos Secuenciales Indexados**
        - Son aquellos archivos (tablas) cuyos registros (filas) están físicamente ordenados secuencialmente según una clave de búsqueda, sobre la cual a la vez existe un índice.
    - También son llamados índices de agrupamiento
    - La clave de búsqueda usualmente es la clave primaria de la tabla.
  - **Índices Secundarios**
    - Son aquellos índices cuya clave de búsqueda no determinan el orden secuencial del archivo ordenado

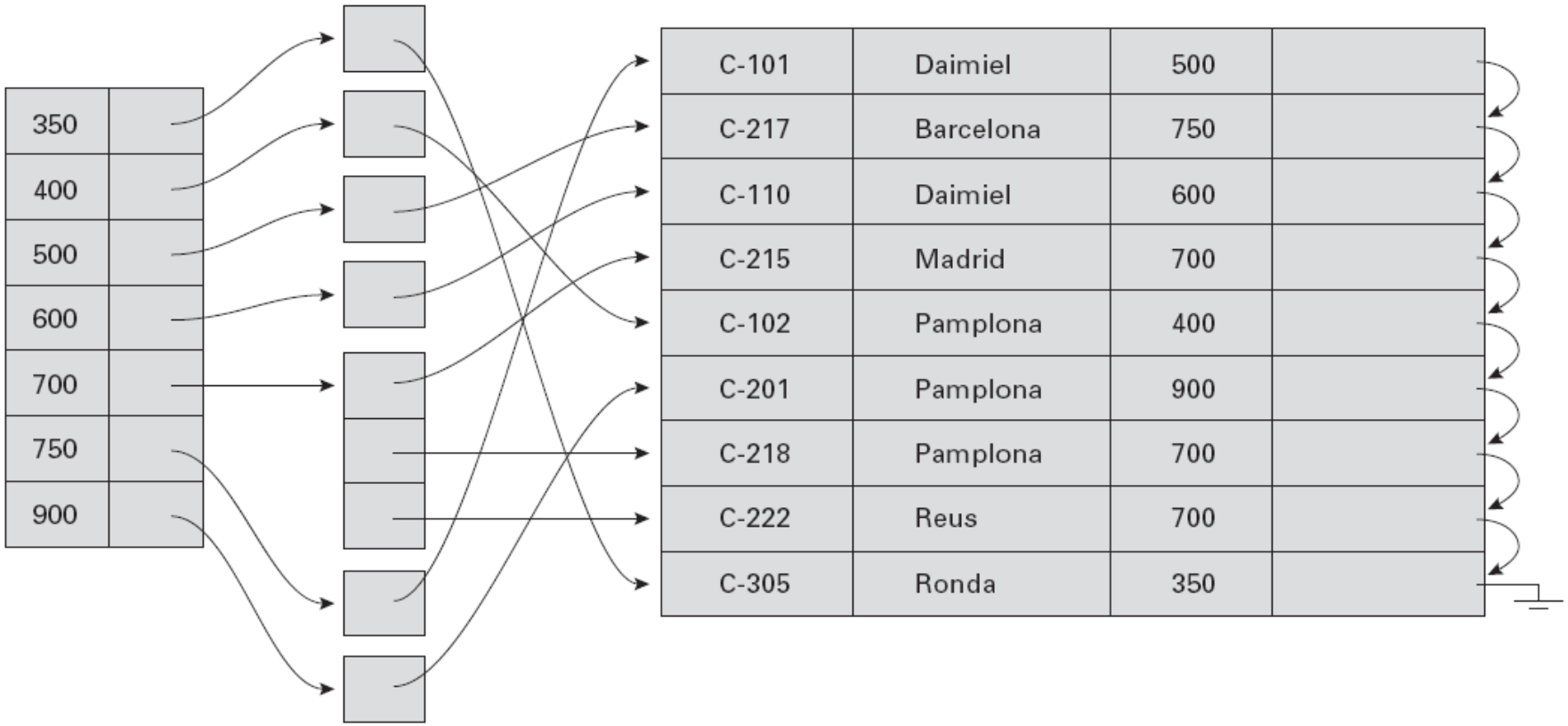
Índices Primarios

- La tabla cuenta es físicamente un archivo secuencial indexado cuyo índice primario tiene como clave de búsqueda a la columna nombre-sucursal



Índices Secundarios

- La tabla cuenta es físicamente un archivo secuencial indexado por el campo o columna nombre-sucursal que tiene un índice secundario definido por el campo o columna saldo.

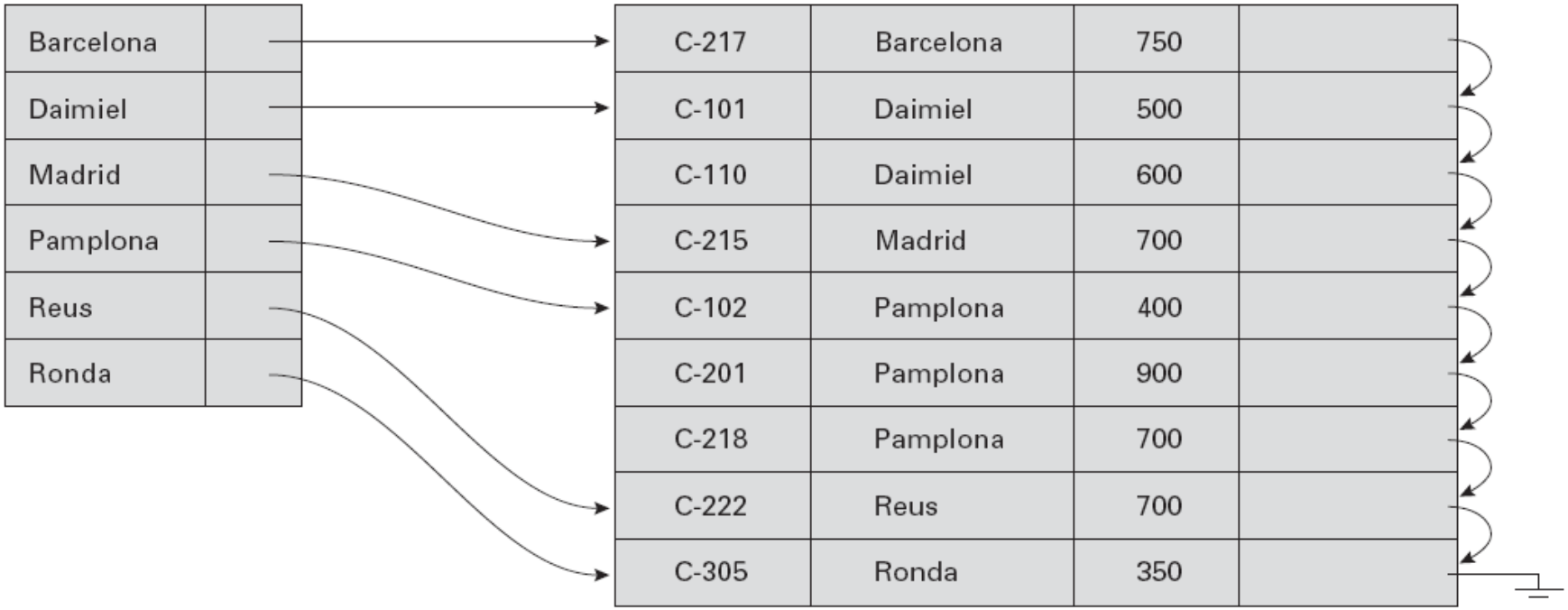


Tipos de Índices ordenados

- Los índices sean primarios o secundarios pueden ser de dos tipos.
  - **Densos**
    - Si poseen un registro índice por cada valor de la clave de búsqueda
  - **Dispersos**
    - Si solo poseen registros índices para algunos valores de la clave de búsqueda.
- Los índices primarios pueden ser tanto densos como dispersos
- Los índices secundarios necesariamente deben ser densos.

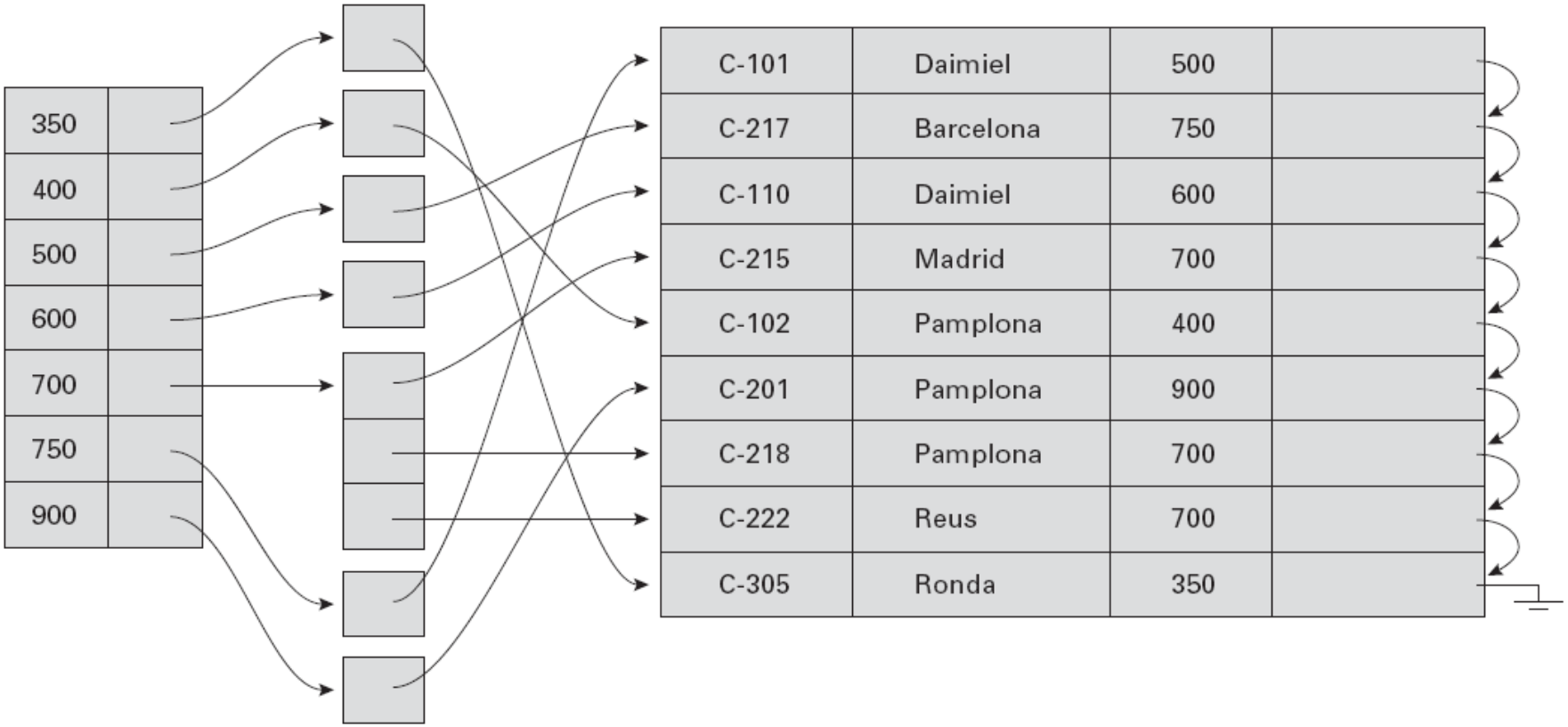
Índices Densos

- En este ejemplo el índice primario definido sobre la columna nombre-sucursal es denso.



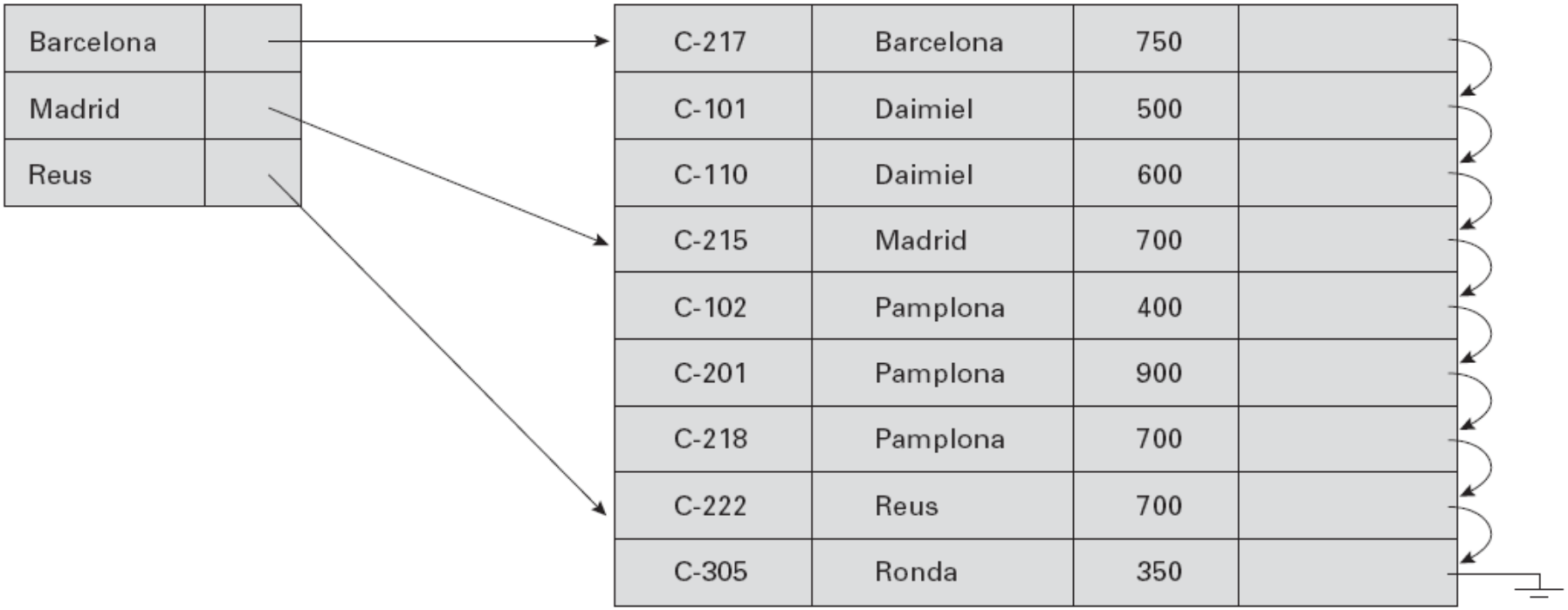
Índices Densos

- En este ejemplo el índice secundario definido sobre la columna saldo es denso.



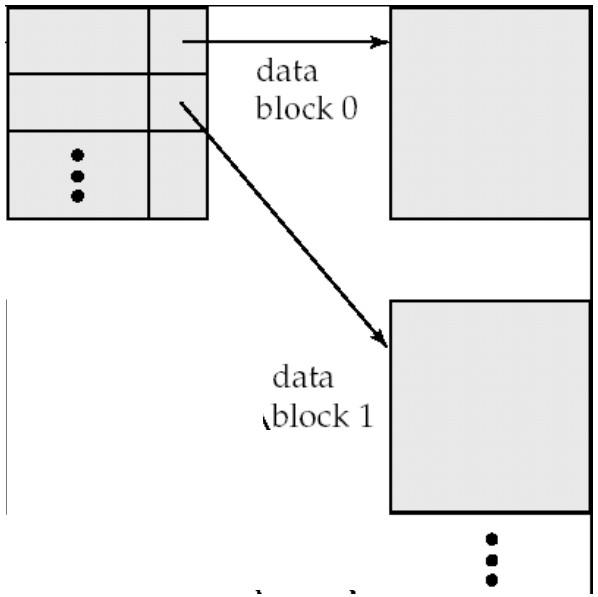
Índices Dispersos

- En este ejemplo el índice primario definido sobre la columna nombre-sucursal es disperso



Índices Dispersos

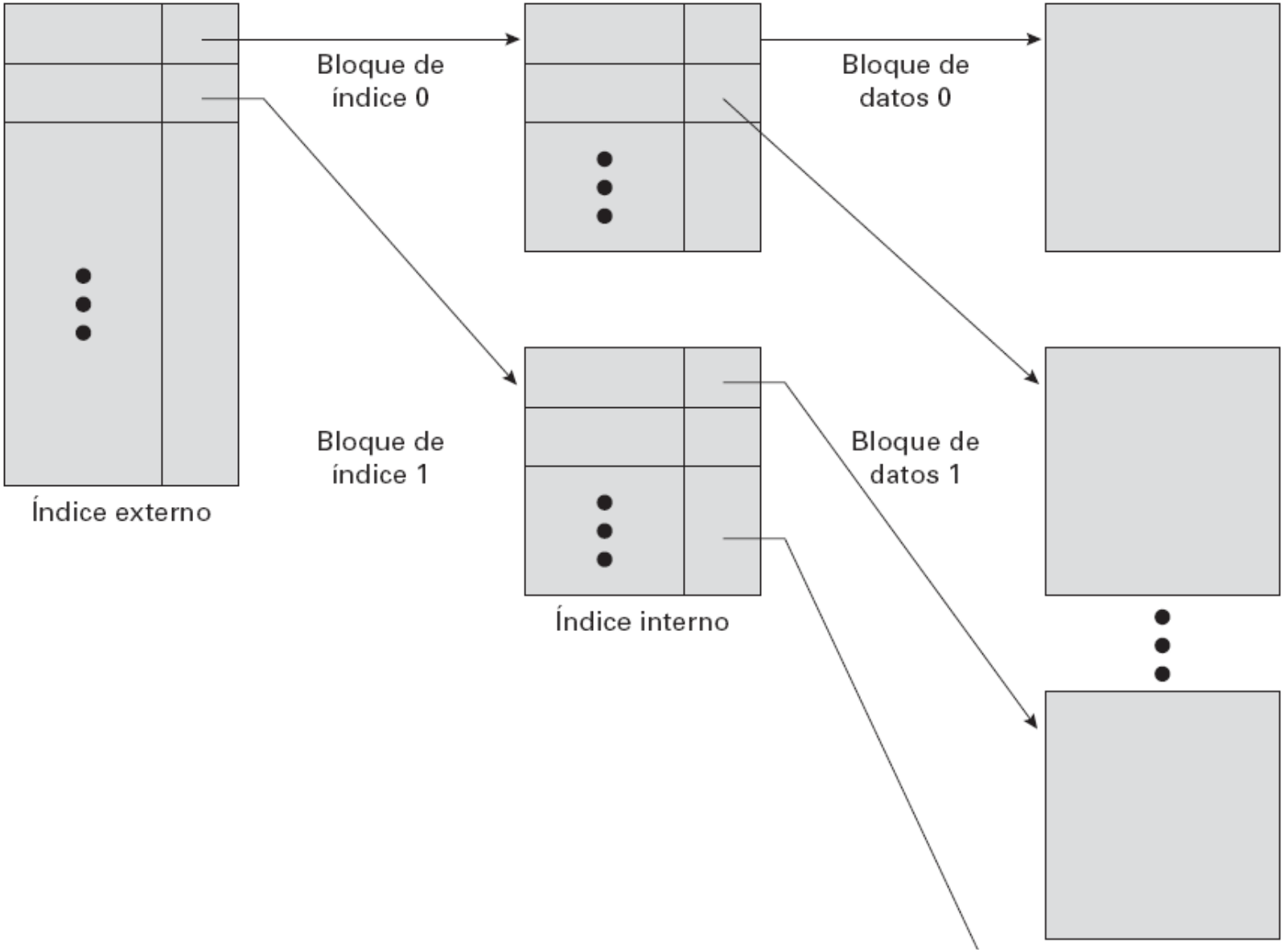
- Los índices dispersos son más apropiados que los índices densos.
  - Requieren menos espacio de almacenamiento
  - Provocan menor sobrecarga en operaciones de inserción y borrado.
- Los índices dispersos pueden derivar en un mayor tiempo de búsqueda comparados con los índices densos.
- Una buena aproximación a la utilización de los índices dispersos es tener una entrada índice por cada bloque del archivo del datos



Índices Multinivel

- La mayoría de las veces la tabla indexada podría ser tan grande tal que los archivos correspondientes a los índices definidos en ella también lo sean.
- Como el archivo índice es un archivo secuencial ordenado es posible volver a construir un índice sobre él.
  - **Índice interno**
    - índice construido sobre el archivo de datos. Puede ser primario o secundario tanto como denso o disperso.
  - **Índice externo**
    - Índice construido sobre un índice interno o externo.
    - Siempre es un índice disperso.
- Es forma de encadenamiento de índices se conoce como **Índices Multinivel**

Índices Multinivel



Inserción en índices

- Sí el índice es denso:
  - Buscar el registro índice con el valor de la clave de búsqueda del registro de datos a insertar
  - Si no existe un registro índice con el valor de la clave de búsqueda, se inserta uno nuevo en la posición adecuada.
  - Si el registro existe:
    - Si se almacena un puntero por cada registro de datos, entonces se agrega un puntero en el cajón asociado al registro índice.
    - Si solo se almacena el puntero al primer registro de datos, no se actualiza el índice.
- Sí el índice es disperso:
  - Buscar registro índice con el mayor valor de clave que sea menor o igual al valor de clave a insertar
  - Insertar el registro de datos en el bloque indicado por el registro índice.
  - Si el bloque se llena, este se divide y se inserta un nuevo bloque de datos, con lo cual se inserta un nuevo registro índice en la posición adecuada, el cual toma el valor de clave del primer registro en el nuevo bloque.

Borrado en índices

- Sí el índice es denso:
  - Buscar el registro índice con el valor de la clave de búsqueda del registro de datos a borrar
  - Si se almacena un puntero porcada registro del archivo de datos, se borra el puntero que indica el registro de datos que se va a borrar.
  - Si solo se almacena el puntero al primer registro y es este es el que se debe borrar, se actualiza el puntero al siguiente registro de datos.
  - Si se eliminan todos los registros de datos con un valor de la clave de búsqueda se elimina el registro índice correspondiente.
- Sí el índice es disperso:
  - Buscar registro índice con el mayor valor de clave menor o igual al valor de clave a insertar
  - Se busca y borrar el registro de datos en el bloque indicado por el registro índice.
  - Si se borran todos los registros de datos con el valor de clave de búsqueda utilizado en el registro índice, este se actualiza con el siguiente valor en el bloque correspondiente
  - Si se eliminan todos los registros de datos en el bloque correspondiente se elimina el registro índice

Archivos Indexados por Árboles B+

- Los archivos secuénciales indexados presentan las siguientes desventajas:
  - El rendimiento decrece a medida que se insertan datos en el archivo de datos, dado que muchos bloques de desbordamiento deben ser creados.
  - Requiere de una reorganización periódica de los datos
- Los archivos indexados por Árboles B+ son una mejor alternativa, que poseen la siguientes ventajas:
  - Los archivos de datos se reorganizan automáticamente solo con pequeños cambios en el archivo de datos en cada inserción o borrado.
  - La perdida de rendimiento no es tan acentuada ante la desorganización del archivo de datos, por lo cual esta su reorganización no es necesaria para mantener el rendimiento global.
- La desventaja:
  - Necesita de un poco más de tiempo para las operaciones de inserción y borrado
  - Requiere de espacio adicional en el archivo de datos.
- Las ventajas de los Árboles B+ sobrepasan a sus desventajas.

Árboles B+

- Un Árbol B+ tiene las siguientes propiedades
  - Todos los caminos desde el nodo raíz a los nodos hojas tienen la misma longitud
  - Cada nodo interno que no es la raíz tiene entre $\lceil n/2 \rceil$ y n hijos.
  - Cada nodo hoja debe tener entre $\lceil (n-1)/2 \rceil$ y n-1 punteros (índice secundario o primario) o registros de datos (índice primario).
  - Casos Especiales
    - Si la raíz es una hoja puede tener entre 0 y n-1 valores
    - Si la raíz no es una hoja debe tener como mínimo 2 hijos

Estructura de un Nodo

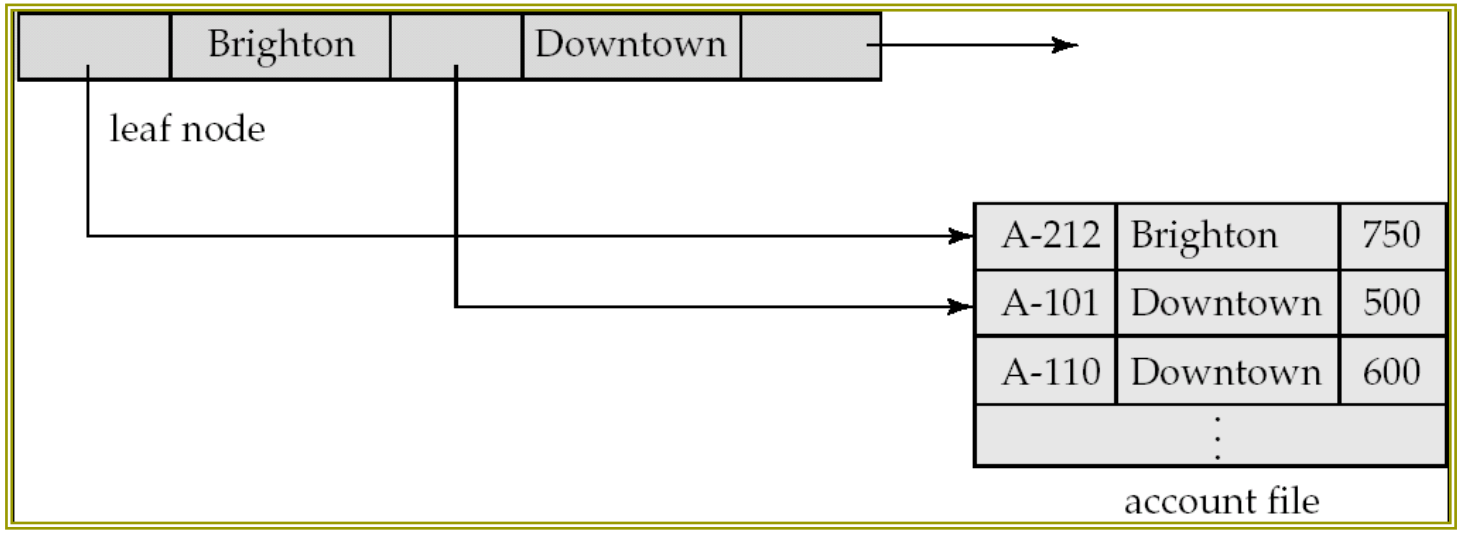
- Nodo Típico

  - K_i son los n-1 valores de la clave de búsqueda
  - P_i son los n punteros a los nodos hijos para lo nodos internos o los punteros a los registros de datos o cajones para los nodos hojas.
- Los valores de la clave de búsqueda están ordenados

$$K_1 < K_2 < K_3 < \ldots < K_{n-1}$$

Nodos Hojas

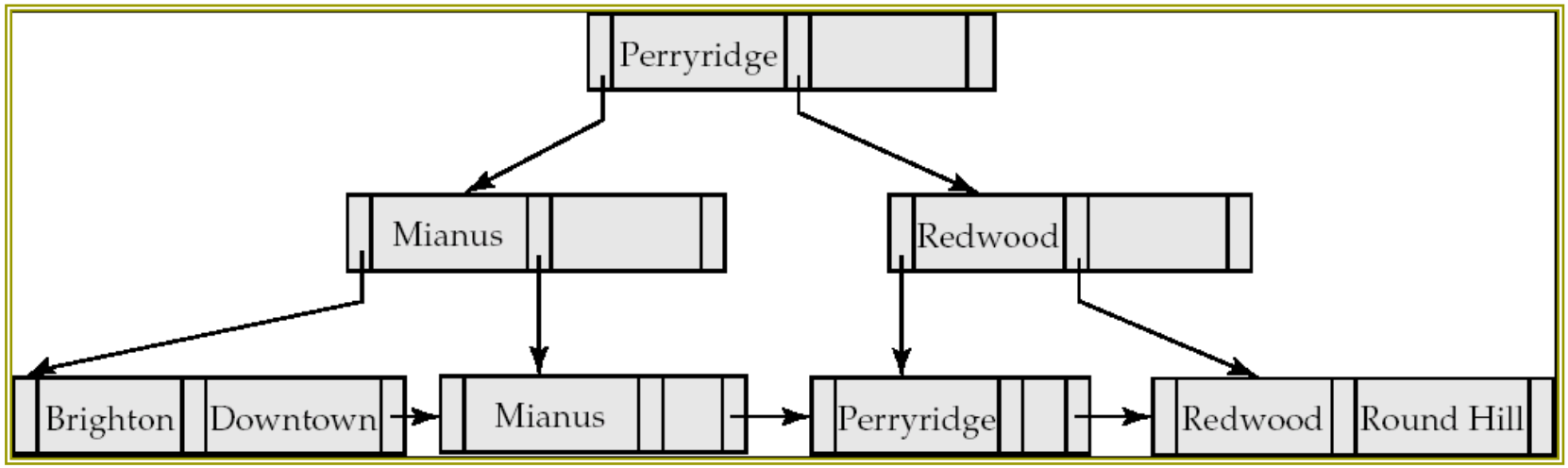
- Propiedades de los nodos hojas
  - Para $i = 1, 2, \ldots, n-1$, el puntero P_i apunta al primer registro de datos o a un cajón con los punteros a los registros de datos.
- Si L_i *y* L_j son nodos hojas con $i < j$, *las claves en* L_i son menores a los valores de clave en L_j
- P_n es un puntero que apunta al siguiente nodo hoja



Nodos Internos

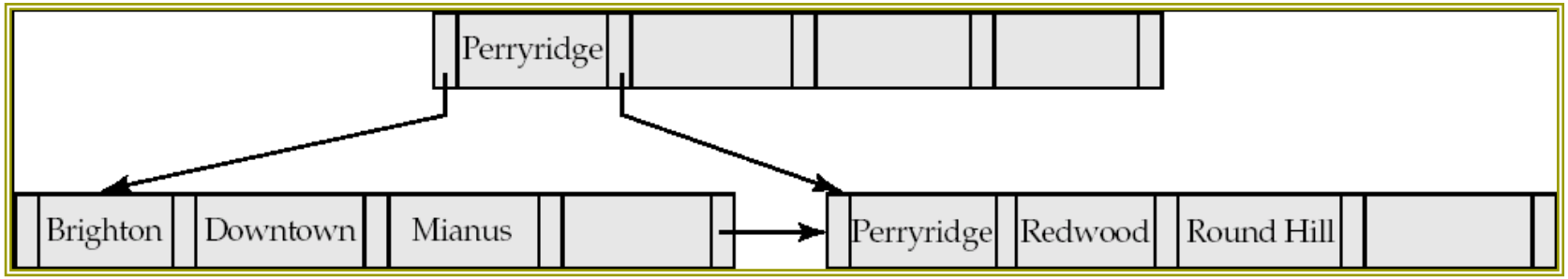
- Para un nodo interno con n punteros:
  - Todas las claves de búsqueda en el subárbol indicado por P_1 son menores a la clave K_1
  - Para $2 \leq i \leq n\text{-}1$, todas las claves de búsqueda en el subárbol determinado por P_i tienen valores mayores o iguales a la clave K_{i-1} y menores que K_i.
  - Todas las claves de búsqueda en el subárbol determinado por P_n tienen valores mayores o iguales a la clave K_{n-1}

Ejemplo de un Árbol B+



- Ejemplo de un Árbol B+ con n=3.
- Cada nodo hoja tiene entre 1 ($\lceil (n-1)/2 \rceil$) y 2 (n-1) valores.
- Cada nodo interno tiene entre 2 ($\lceil n/2 \rceil$) y 3 (n) punteros a nodos hijos.
- La raíz al no ser hoja tiene como mínimo 2 hijos

Ejemplo de un Árbol B+



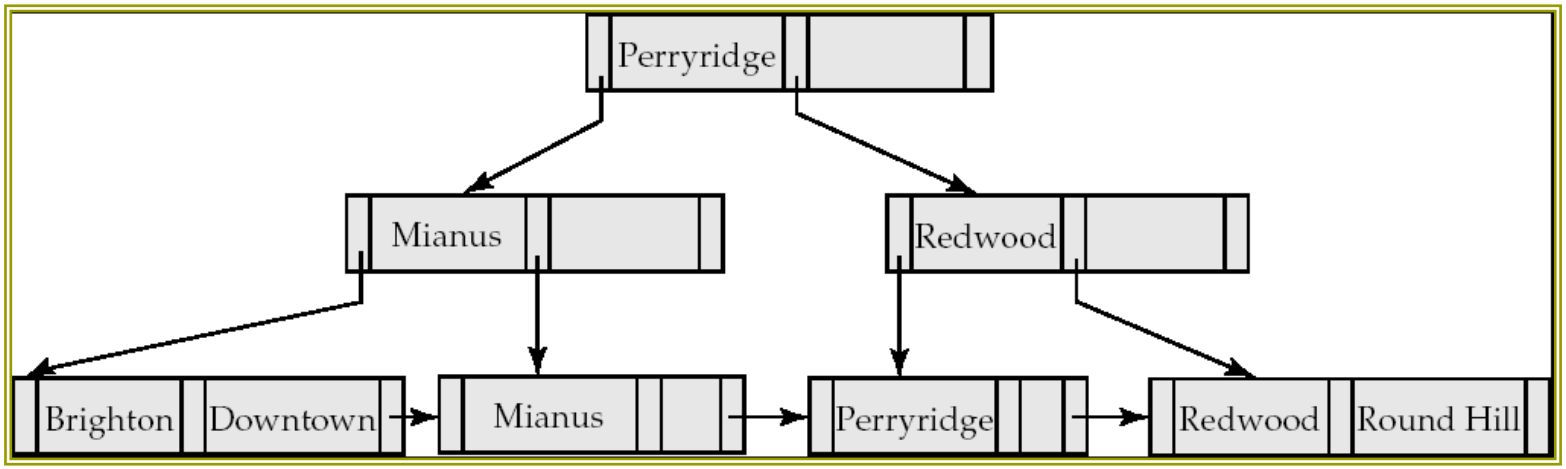
- Ejemplo de un Árbol B+ con n=5.
- Cada nodo hoja tiene entre 2 ($\lceil (n-1)/2 \rceil$) y 4 (n-1) valores.
- La raíz al no ser hoja tiene como mínimo 2 hijos

Índices como Árboles B+

- Cada Nodo de un Árbol B+ puede ser un bloque.
- Los punteros de los nodos apuntan a otro bloque que no necesariamente debe estar contiguo en el disco.
- Los niveles internos forman una jerarquía de índices dispersos multinivel
- Los árboles B+ tienen muy pocos niveles
  - Para un total de K valores diferentes de la clave de búsqueda la altura del árbol esta dada según $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$
  - La búsquedas son entonces muy eficientes.
- Inserciones y borrados en el archivo de datos y la actualización del índice pueden realizarse con muy pocos accesos al disco.

Búsqueda en un Árbol B+

- Buscar el o los registros con valor de clave k.
  1. *N = Raíz*
  2. Repetir hasta que *N* sea una hoja
     1. Buscar en *N* la menor entrada K_i que sea mayor a *k.*
     2. Si la entrada K_i *existe, entonces* $N = P_i$
     3. Sino entonces $k \geq K_{n-1}$, y se hace $N = P_n$
  3. Si en la hoja existe una clave $K_i = k$, seguir el puntero P_i para ubicar el registro de datos o el cajón deseado.
  4. Sino, si en la hoja existe una clave $K_i = k$, no existe un registro de datos que posea dicho valor de la clave de búsqueda.

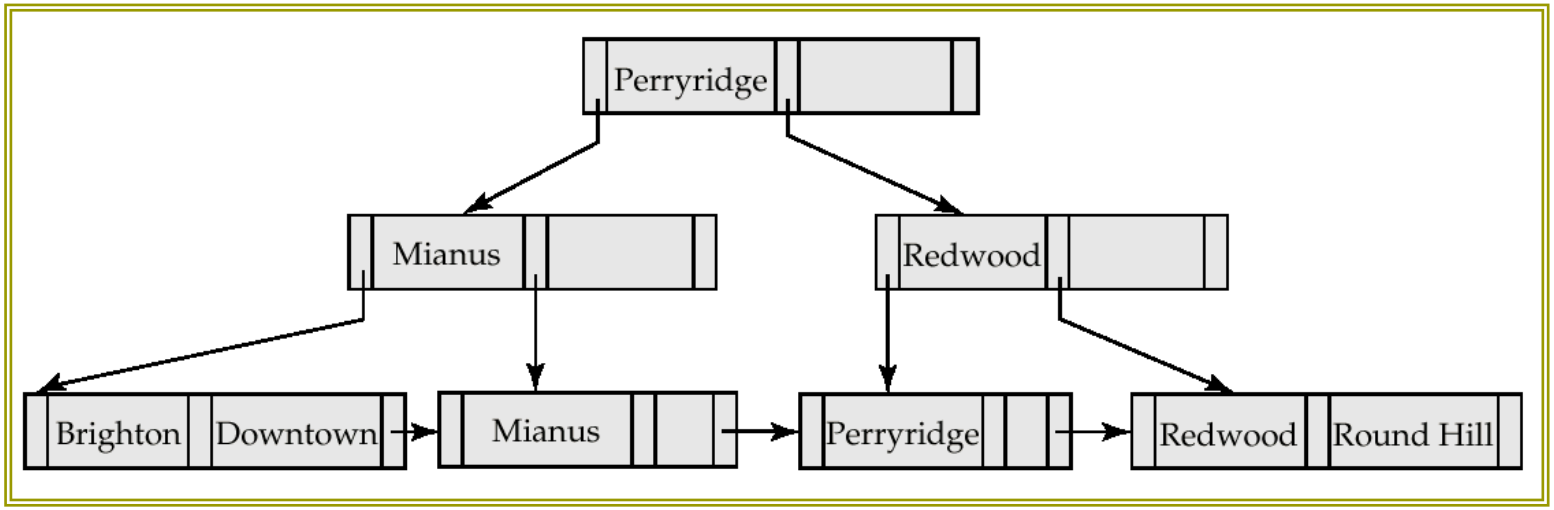


Inserción en un Árbol B+

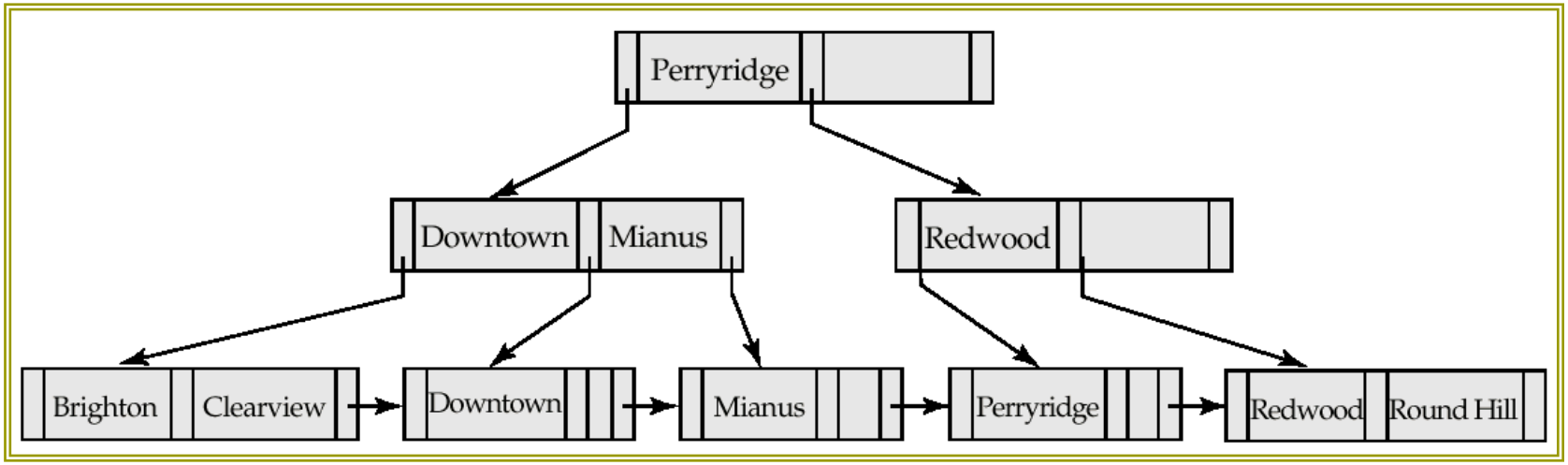
1. Buscar el nodo hoja en el cual el valor de la clave K del registro a insertar debe aparecer
2. Si el valor de clave existe
   1. Agregar un registro al archivo o un puntero al cajón.
3. Si el valor de la clave no existe
   1. Agregar un registro al archivo o un puntero al cajón.
   2. Si hay lugar en el nodo hoja, agregar un nuevo par clave puntero (K, P) en el nodo.
   3. Sino hay lugar en el nodo, el nodo debe ser dividido.
      1. Ordenar los n pares clave, puntero (incluido el que se inserta)
      2. Dejar los primeros $\lceil n/2 \rceil$ pares en el nodo original y el resto en el nuevo nodo S.
      3. Sea K el menor valor de clave en el nuevo nodo S insertar el nuevo par (S,K) en el padre P del nodo original.
      4. Repetir los pasos 1 a 3 si P esta lleno, hasta la raíz.

Inserción en un Árbol B+

Árbol B+ antes de la inserción de “Clearview”



Árbol B+ luego de la inserción de “Clearview”

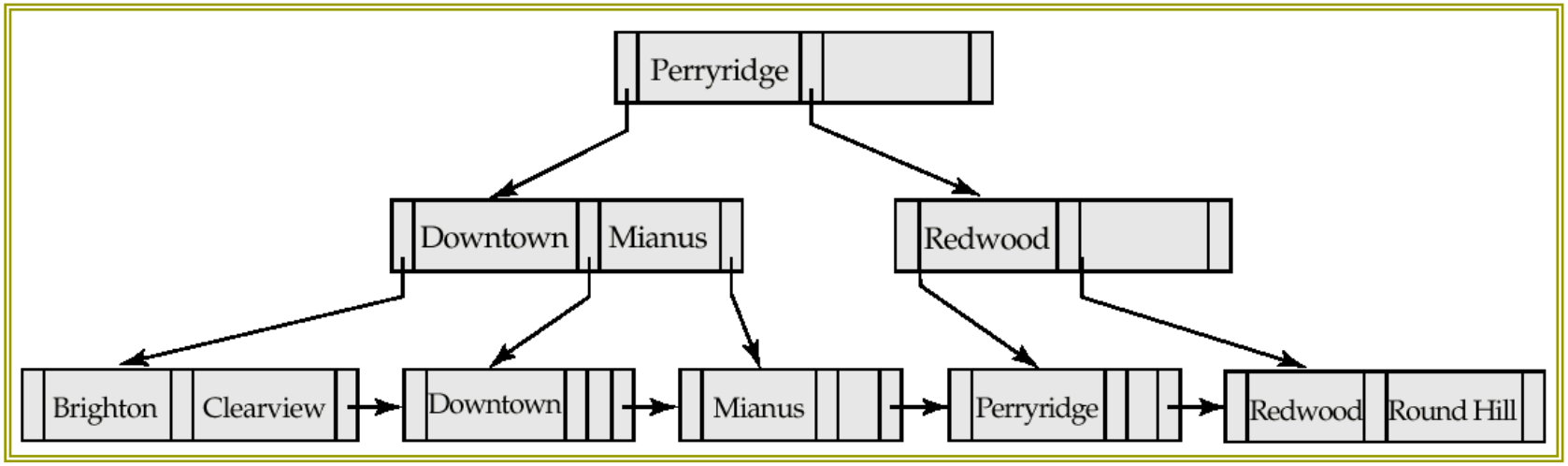


Borrado en un Árbol B+

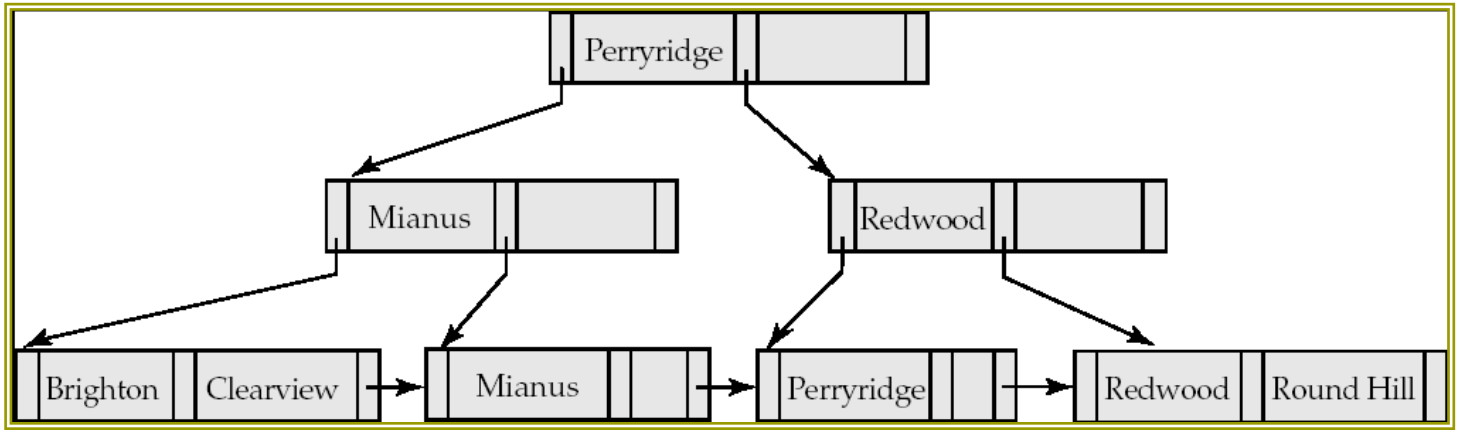
1. Buscar el puntero P correspondiente a la clave K buscada y remover el registro del archivo de datos o el puntero del cajón si existe.
2. Remover el par (K,P) del nodo hoja si corresponde.
3. Si el nodo queda sin suficientes valores y sus entradas pueden caber completamente en uno de sus nodos vecinos, entonces unir los nodos
    1. Insertar todos los pares clave punteros en el nodo de la izquierda y borrar el nodo.
    2. Borrar el par clave puntero correspondiente al nodo borrado en el nodo padre.
4. Si el nodo queda sin suficientes valores, pero no puede ser combinado con sus nodos vecinos, entonces:.
    1. Redistribuir los pares clave-puntero con el vecino que tenga mayor cantidad de claves
    2. Actualizar el correspondiente par clave-puntero en el en el nodo padre correspondiente.
5. Continuar con la actualización hasta que todos los nodos queden con $\lceil n/2 \rceil$ punteros
6. Si la raíz queda con un solo puntero, hacer su único hijo la nueva raíz.

Borrado en un Árbol B+

Antes de Borrar la entrada "Downtown"



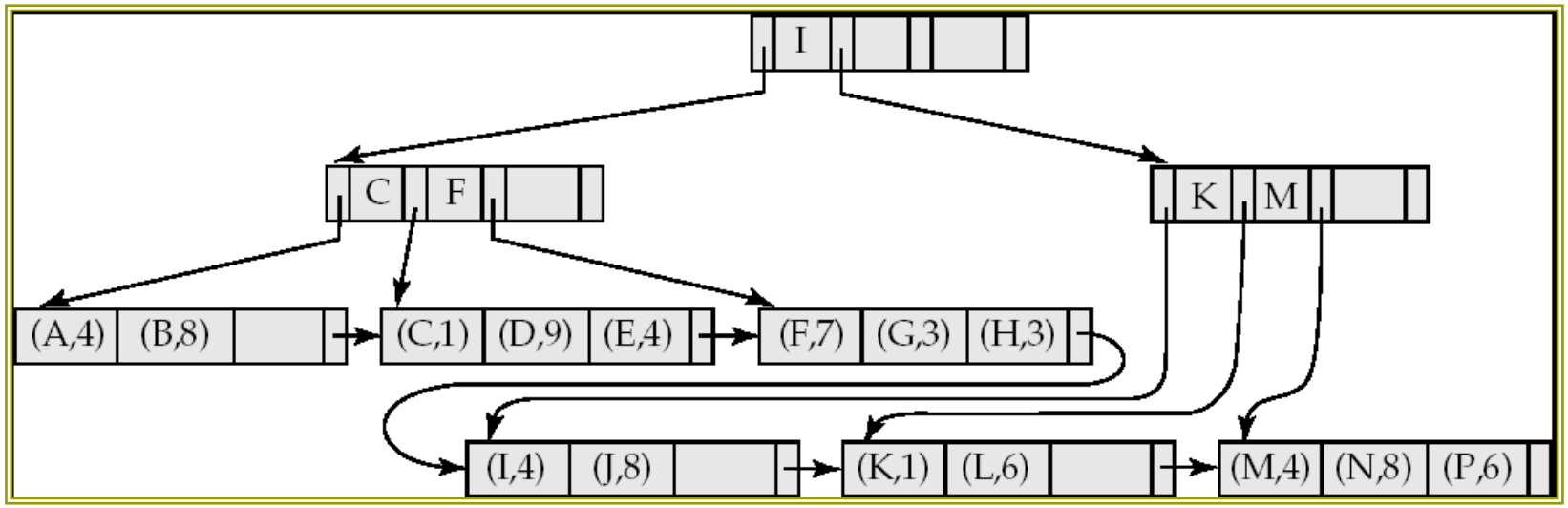
Luego de Borrar la entrada "Downtown"



Organización de Archivos con Árboles B+

- Los árboles B+ pueden resolver la degradación de los archivos de datos
- En una organización de archivo de árbol B+ las hojas del árbol almacenan registros de datos en vez de punteros.
- Las hojas del árbol aun requieren que estén medio llenas
  - Dado que los registros son mas grandes que los punteros un registro hoja solo puede almacenar menos de n registros.
- Los algoritmos de inserción y borrado son ejecutados igual que el los índices B+.

Organización de Archivos con Árboles B+

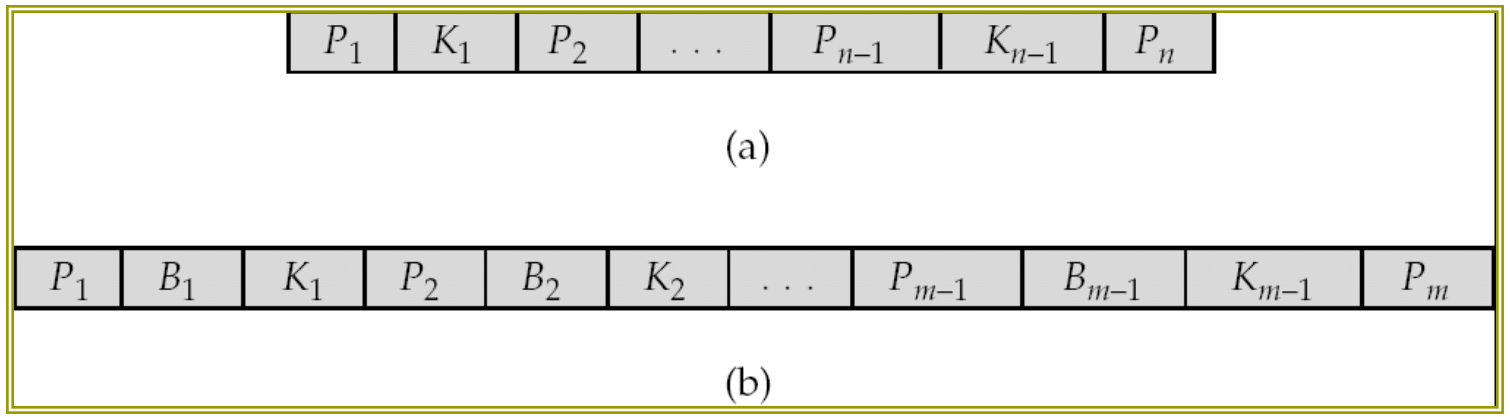


Ejemplo de Organización de Archivo con Árbol B+

- Buena utilización del espacio, aun aunque los registros utilicen más espacio que los punteros.
- Para mejorar la utilización del espacio más nodos hermanos deben ser divididos o mezclados
  - La redistribución debe realizarse cuando los nodos estén ocupados en menos de $\lfloor 2/3 \rfloor$ de su capacidad

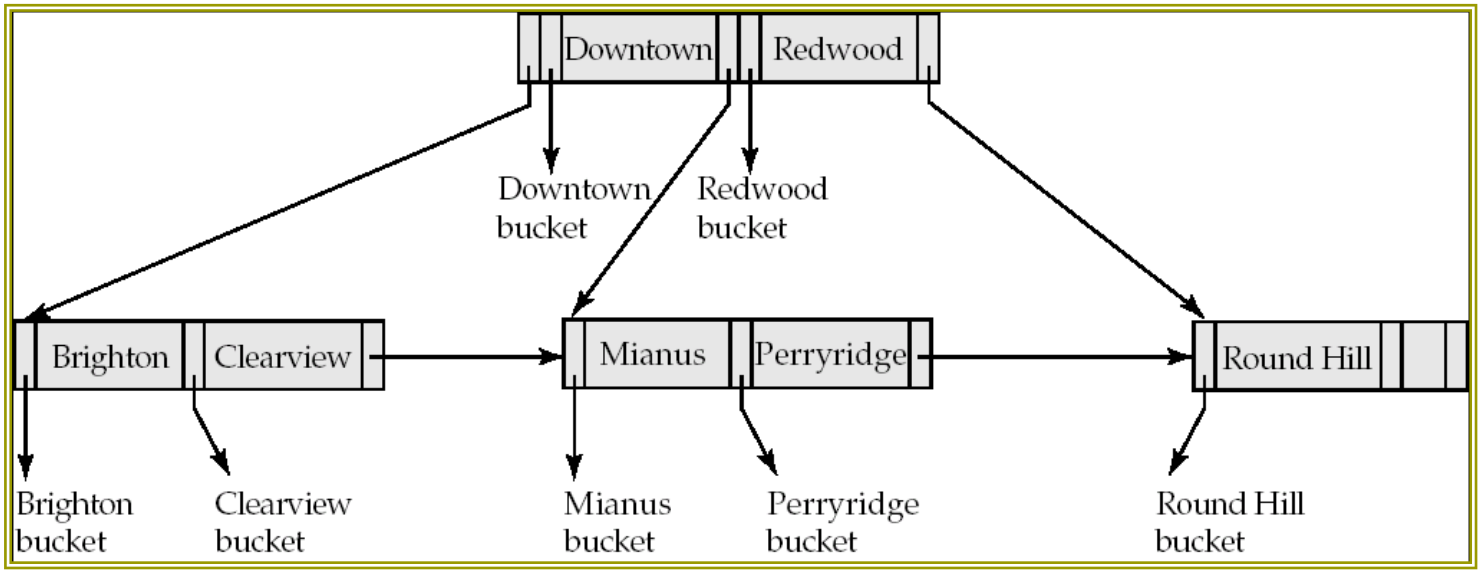
Índices de árbol B

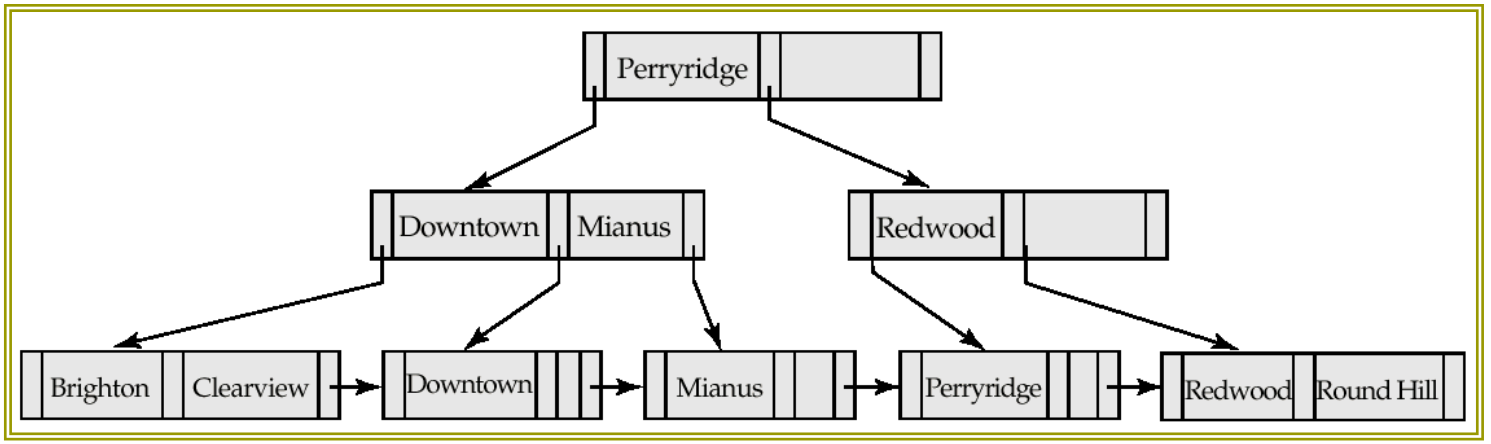
- Similares a los árboles B+, sin embargo los valores de la clave de búsqueda no se repiten en el árbol, por lo que se elimina el uso redundante del espacio.
- Los nodos internos tienen un puntero adicional para referenciar al bloque de datos o cajón asociado al valor de la clave de búsqueda.
- Los punteros de los nodos hojas apuntan al cajón o bloque de datos como en los árboles B+.



Índices de árbol B

- Un árbol B y un árbol B+ para los mismos datos





Índices de árbol B

- Ventajas:
  - Menos cantidad de nodos que su correspondiente árbol B+
  - Las búsquedas de algunas claves son más rápidas debido a que no se alcanza el nivel de las hojas.
- Desventajas
  - Solo un pequeño numero de valores de la clave de búsquedas son encontrados antes.
  - Los nodos hojas requieren más espacio reduciendo la amplitud de direccionamiento.
  - Inserciones y borrados son más costosos y complicados.
  - La implementación es más difícil que los árboles B+.

Organización por Asociación

- Un cajón es una unidad de almacenamiento de registro de datos, típicamente de tamaño igual a un bloque.
- En una **Organización de Archivos por Asociación** se obtiene la dirección del cajón a través de la función de dispersión o hash sobre el valor de la clave de búsqueda.
- La Función Hash *h* es una función de todos los valores de claves sobre el conjunto de cajones.
- La Función Hash se aplica para insertar, borrar o buscar un registro de datos.
- Registros con diferentes valores de clave de búsqueda pueden ser ubicados en el mismo cajón, Por ello un cajón debe ser explorado secuencialmente para localizar un registro de datos.

Asociación Estática

- Un esquema de asociación estática es aquel en donde el número de cajones es constante.

Cajón 0

Cajón 1

Cajón 2

Cajón 3

C-217	Barcelona	750
C-101	Daimiel	500

Cajón 4

C-222	Reus	700

Cajón 5

C-102	Pamplona	400
C-201	Pamplona	900
C-218	Pamplona	700

Cajón 6

Cajón 7

C-215	Madrid	700

Cajón 8

C-305	Ronda	350
C-110	Daimiel	600

Cajón 9

Función Hash

- Una mala Función Hash es aquella que mapea todos los registros a un mismo cajón, haciendo que el tiempo de la operaciones proporcional al numero de registros por el coste de acceso secuencial.
- Una Función Hash ideal debe
  - **Uniforme:**
    - Considerando todos los valores de la clave de búsqueda, cada cajón es asignado con el mismo número de claves de búsqueda.
  - **Aleatoria**:
    - Cada cajón debe recibir aproximadamente el mismo numero de registros sin importar cual sea la distribución de aparición de los valores de clave dada.
- Las Funciones Hash por lo general se basan en la representación binaria de la clave de búsqueda.
  - Para una clave de búsqueda del tipo cadena de caracteres, la función hash podría retornar el la suma de la representación binaria de sus caracteres modulo el número de cajones.

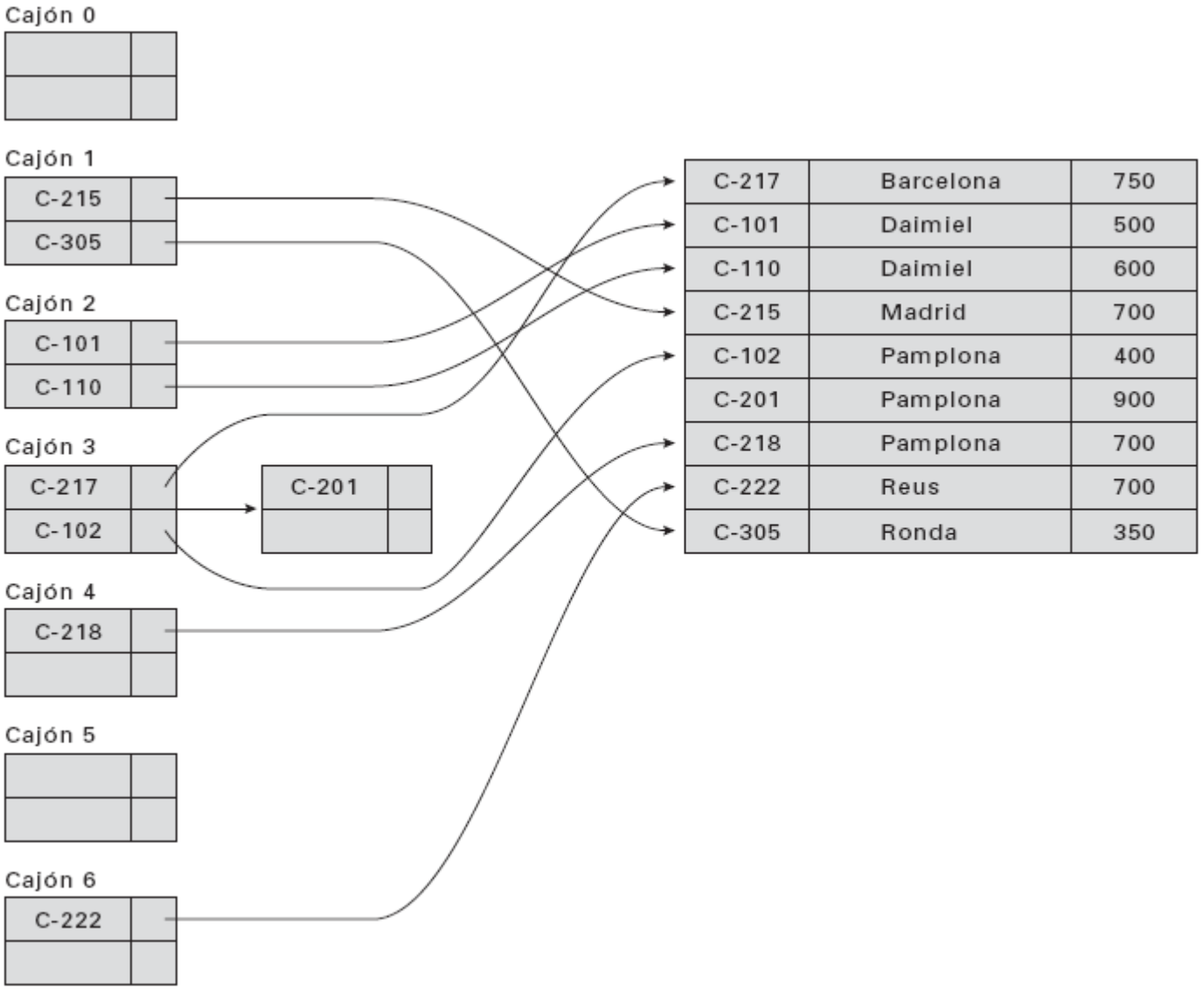
Cajones de desbordamiento

- El desbordamiento de los cajones puede ocurrir por:
  - Insuficiencia de cajones
  - Mala distribución de los registros.
    - Múltiples registros con la misma clave de búsqueda
    - La función hash no es uniforme
- El desbordamiento de los cajones no puede ser evitado, cuando un cajón se llena, los registro subsecuentes son insertados en cajones de desbordamiento.
- Dos enfoques:
  - Cada cajón tiene sus cajones de desbordamientos. Los cuales están enlazados. **Hashing Cerrado.**
  - El desbordamiento de un cajón se maneja insertando los registros desbordados en el cajón siguiente. **Hashing Abierto**

Índices Hash

- Las estrategia Hash no solo puede ser utilizada para organizar un archivo de datos, sino también para estructurar un archivo índice.
- Un índice hash organiza los registros índices como archivos de asociación.
- Es posible construir un índice hash sobre cualquier columna de la tabla y sus entradas siempre serán registros índices organizados en cajones. Cada registro es un par clave-puntero

Índices Hash



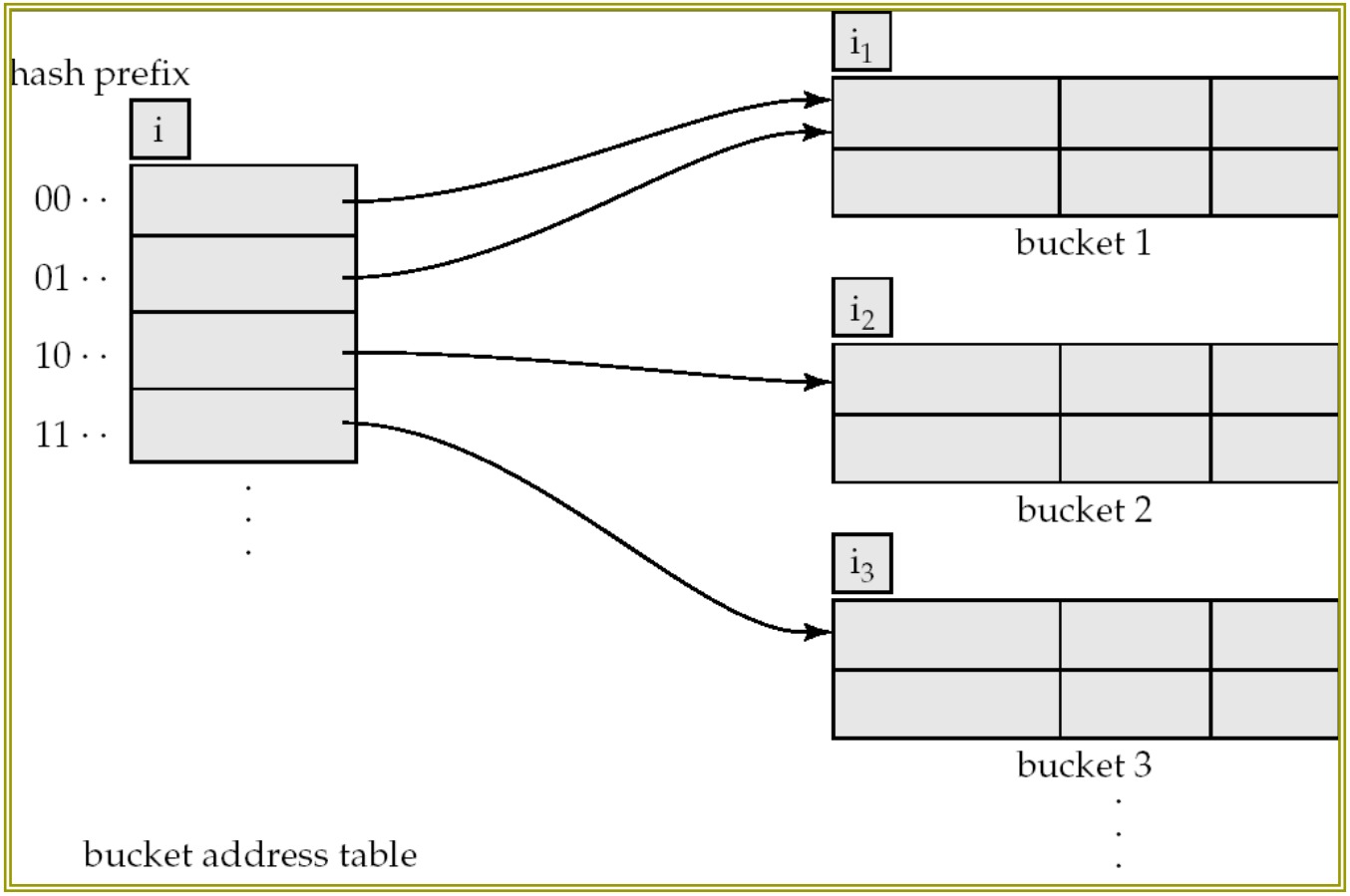
Deficiencias de la Asociación Estática

- Debido a que la cantidad de datos pueden aumentar o disminuir con el tiempo una estructura de asociación estática puede presentar las siguientes deficiencias:
  - En un instante dado el número de cajones puede ser muy pequeño con respecto a la cantidad de datos, entonces el rendimiento se degrada debido al excesivo desbordamiento de los datos.
  - En otro instante el número de cajones es muy grande con respecto a la cantidad de datos, por lo que existe mucho espacio desperdiciado.
- Una solución es reorganizar periódicamente los datos aumentando o disminuyendo el número de cajones.
  - Operación con coste proporcional a la cantidad de datos
  - Debe realizarse cuando la carga del sistema sea baja.

Asociación dinámica

- Eficiente para casos en que la cantidad de datos aumenta y disminuye en gran medida a través de tiempo.
- Permite que la cantidad de cajones sea modificada dinámicamente.
- Asociación extensible (**Extendable Hashing**):
  - La función hash debe generar valores en un rango grande. Típicamente enteros de 32 bits.
  - Se utiliza un prefijo de la representación binaria de los valores hash para indexar los cajones.
  - Siendo **i** la cantidad de bits del prefijo aplicado:
    - Se tiene una tabla de direccionamiento que permite indexar hasta 2^**i** cajones.
  - El valor de **i** aumenta o disminuye con relación a la cantidad de datos, provocando que los cajones sean divididos para aumentar la capacidad o combinados para recuperar espacio
  - Varias entradas de la tabla de direccionamiento pueden apuntar al mismo cajón.

Asociación Extensible



En esta estructura, $i_2 = i_3 = i$, mientras que $i_1 = i - 1$

Asociación Extensible

- Cada cajón ***j*** almacena los registros de datos para los cuales su valor hash coincide en los primeros i_j bits.
- Para localizar los registros de datos de cierta clave **K** se debe:
  - Computar el valor h(**K**) = **X**.
  - Usar los primeros **i** bits de **X** para ubicar la entrada correspondiente en la tabla de direcciones.
  - Ir al cajón apuntado por la entrada de la tabla de direcciones.
- Para insertar un registro de datos con clave K se debe:
  - Seguir el mismo procedimiento que en la búsqueda para localizar el cajón correspondiente.
  - Si hay espacio, insertar el registro de datos en el cajón.
  - Sino, aumentar el número de cajones para aumentar el espacio y volver a intentar la inserción del registro.

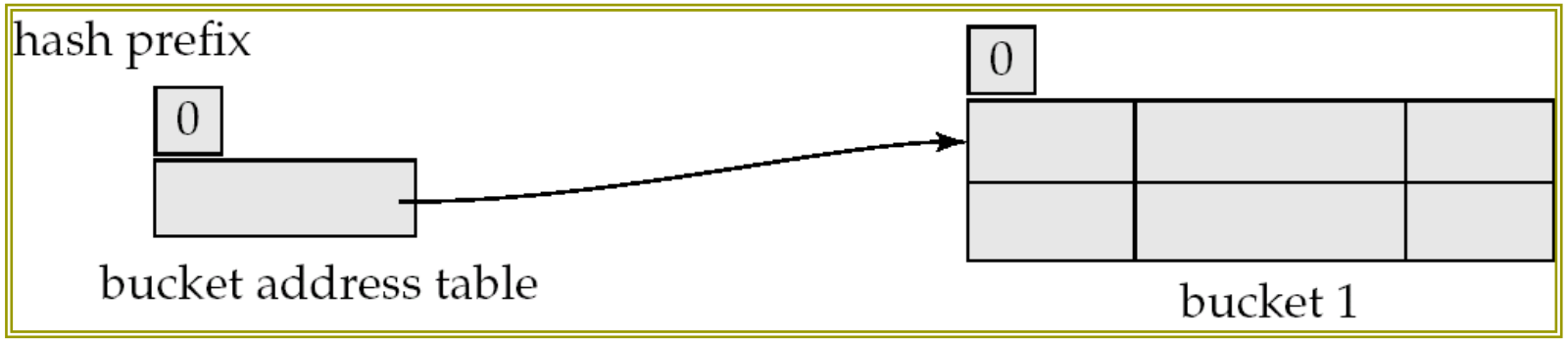
Asociación Extensible

- Cuando ocurre que para insertar un registro con clave **K** se debe aumentar el número de cajones se procede como sigue:
  - Si $i_j < i$ se tiene que más de un puntero en la tabla de direcciones apunta al cajón ***j***, entonces:
    - Se añade un nuevo cajón ***z*** y se hace $i_j = i_z = i_j + 1$
    - Se hace que la segunda mitad de los punteros de la tabla de direcciones que apuntan al cajón ***j*** apunten al cajón ***z***.
    - Recálcular el hash de los registros en el cajón ***j*** para determinar su nuevo cajón en base al valor del nuevo bit considerado, valor 0 queda en el cajón ***j***, valor 1 va al cajón ***z***
    - Insertar el nuevo registro con ***K***.
  - Si $i_j = i$ se tiene que un solo puntero en la tabla de direcciones apunta al cajón ***j***, entonces:
    - Si se ha alcanzado el limite de bits de la función hash y solo queda añadir un cajón de desbordamiento para el cajón.
    - Sino,
      - Se incrementa el valor de ***i,*** duplicándose el tamaño de la tabla de direcciones.
      - Actualizar la tabla de direcciones para que cada dos entradas consecutivas apunten al mismo cajón correspondiente.
      - Tratar la inserción del nuevo registro como en el caso anterior ya que ahora $i > i_j$ para el cajón ***j*** en el que debe ir el nuevo registro.

Asociación Extensible

A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

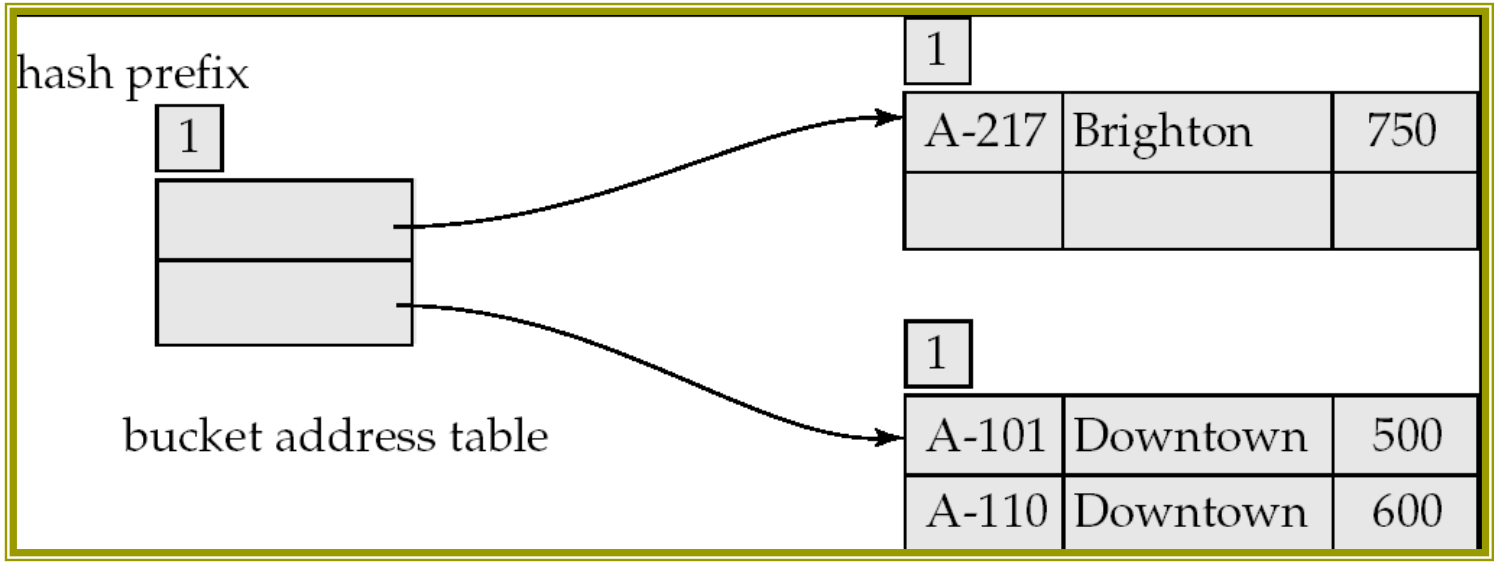
branch_name	h(*branch_name*)
Brighton	0010 1101 1111 1011 0010 1100 0011 0000
Downtown	1010 0011 1010 0000 1100 0110 1001 1111
Mianus	1100 0111 1110 1101 1011 1111 0011 1010
Perryridge	1111 0001 0010 0100 1001 0011 0110 1101
Redwood	0011 0101 1010 0110 1100 1001 1110 1011
Round Hill	1101 1000 0011 1111 1001 1100 0000 0001



Estructura Inicial, Tamaño de los cajones = 2

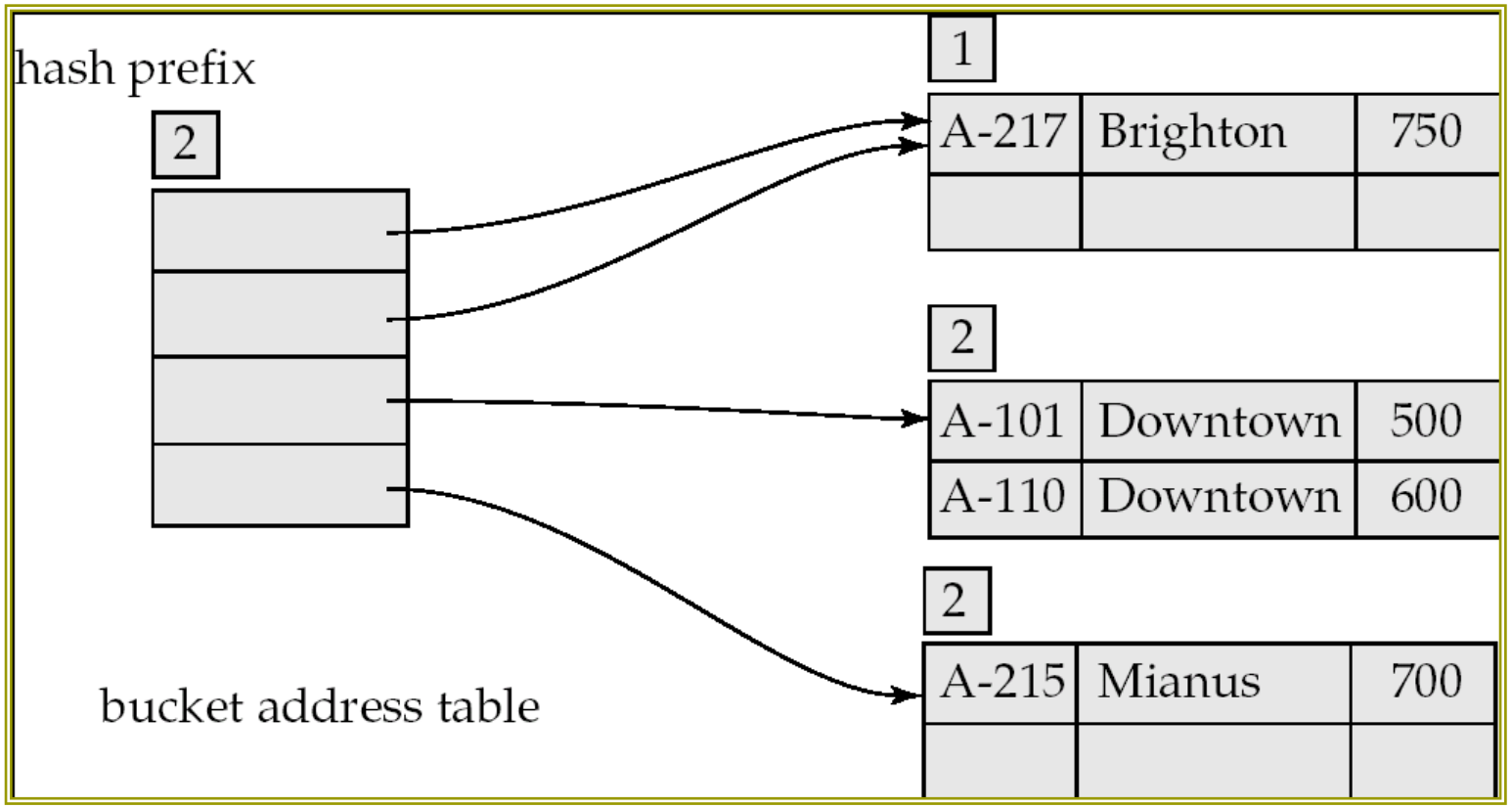
Asociación Extensible

- Estructura luego de insertar los registros correspondientes a Brighton y Downtown



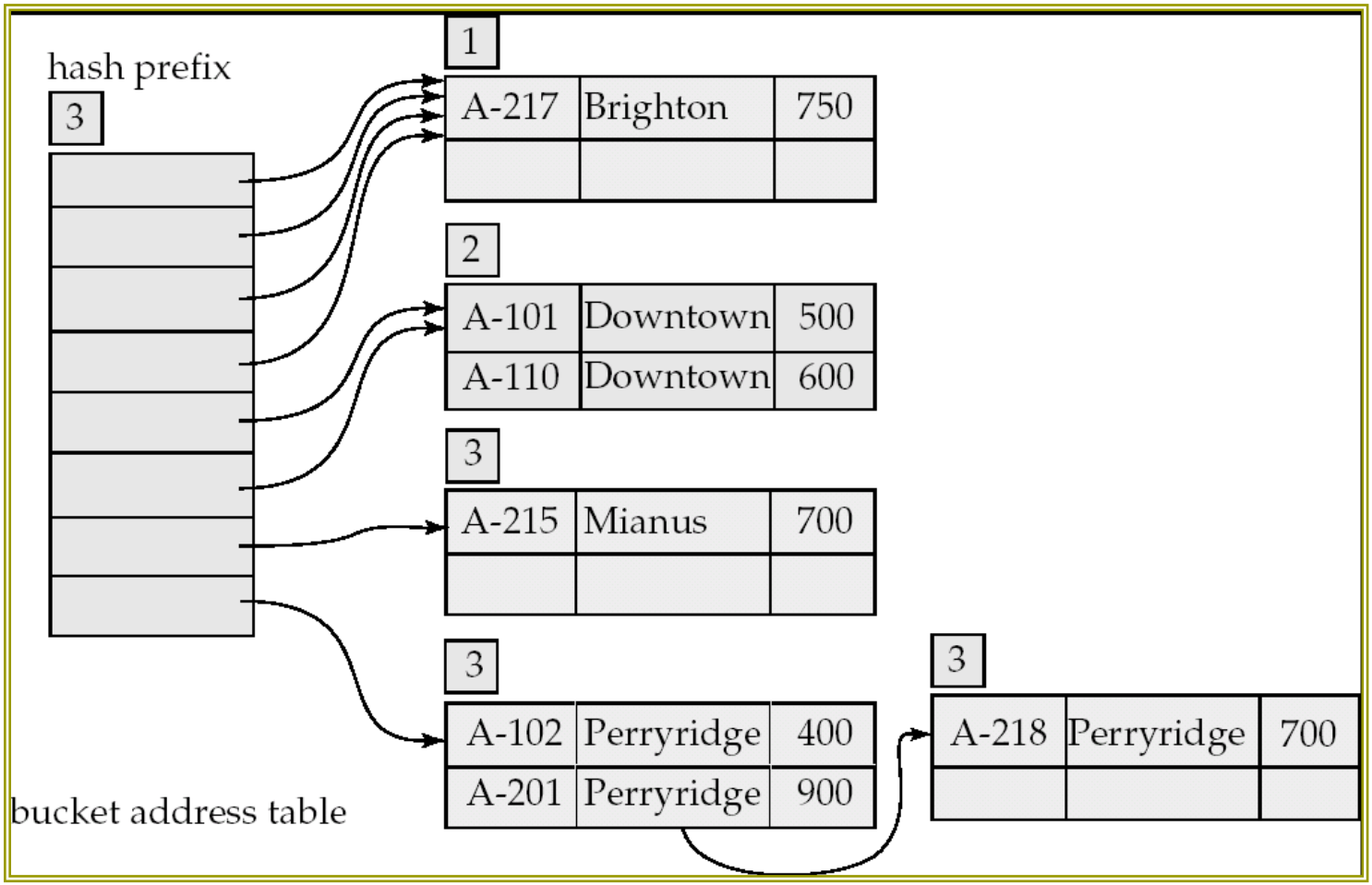
Asociación Extensible

- Estructura luego de insertar el registro correspondientes a Mianus



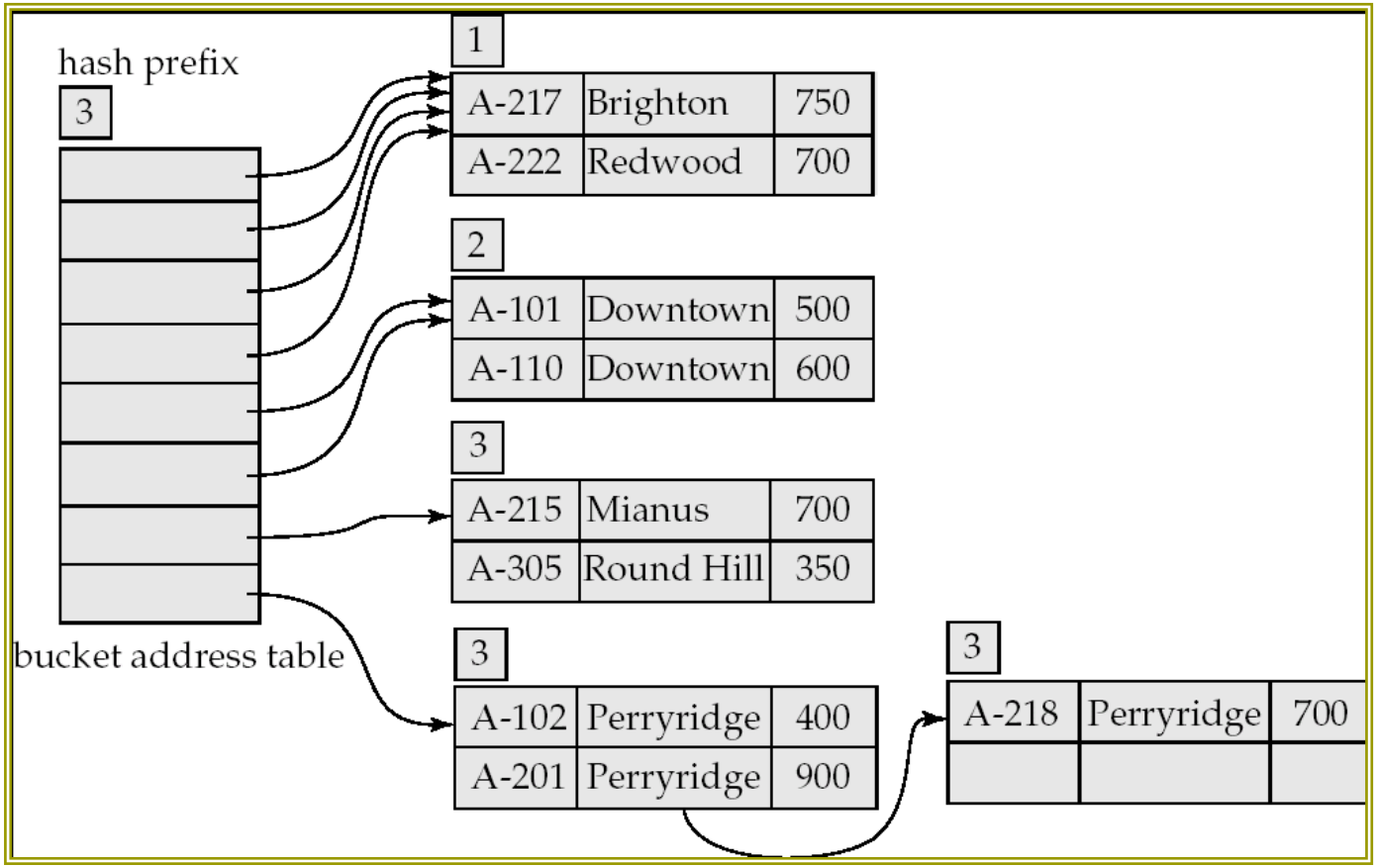
Asociación Extensible

- Estructura luego de insertar los registros correspondientes a Perryridge



Asociación Extensible

- Estructura luego de insertar todos los registros



Asociación Extensible

- Para borrar un registro:
  - Se debe buscar el cajón que lo contiene y removerlo del mismo.
  - El cajón debe ser removido al vaciarse, actualizando apropiadamente su entrada en la tabla de direcciones.
  - La fusión de cajones también puede ser realizada si sus entradas pueden caber en un solo cajón.
    - Solo se pueden funcionar cajones con el mismo valor de tamaño de prefijo i_j y que además solo difieren en el ultimo bit del prefijo
  - La reducción de la tabla de direcciones también pude ser efectuada.
    - Esta operación es costosa y solo debería realizarse si el numero de cajones es considerablemente menor que el tamaño de la tabla.

Índices Bitmaps

- Los índices bitmaps son un tipo especial de índices diseñados para consultar eficientemente en múltiples claves.
- Se asumen que los registros de datos están numerados secuencialmente.
  - Dado un numero es fácil recuperar ese registro.
- Aplicable a atributos que pueden ser descompuestos en una serie de valores distintos
  - Genero, País, Estado…
  - Ej. Nivel de Ingresos (0-999, 1000-9999,10000-50000, …)

Índices Bitmaps

- En su forma simple un Índice Bitmap tiene un array de bits para cada valor del atributo del índice.
  - Cada array tiene tantos bits como registros existan
  - En un array en particular, el bit **n**:
    - Toma valor 1, si el registro **n** tiene el valor de clave correspondiente al array
    - Toma valor 0 en caso contrario.

record number	*name*	*gender*	*address*	*income_level*
0	John	m	Perryridge	L1
1	Diana	f	Brooklyn	L2
2	Mary	f	Jonestown	L1
3	Peter	m	Brooklyn	L4
4	Kathy	f	Perryridge	L3

Bitmaps for *gender*

m	10010
f	01101

Bitmaps for *income_level*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Índices Bitmaps

- Los índices bitmap son útiles tanto para consultas de sobre un atributo o sobre varios atributos, siempre y cuando existe un índice bitmap en cada atributo.
- Consultas sobre los atributos se resuelven aplicando operaciones a nivel de bit:
  - Una condición OR, se obtiene haciendo la operación OR bit a bit entre los bitmaps.
  - Una condición AND, se obtiene haciendo la operación AND bit a bit…
  - La negación se obtiene complementando el bitmap.

record number	*name*	*gender*	*address*	*income_level*
0	John	m	Perryridge	L1
1	Diana	f	Brooklyn	L2
2	Mary	f	Jonestown	L1
3	Peter	m	Brooklyn	L4
4	Kathy	f	Perryridge	L3

Bitmaps for *gender*

m	10010
f	01101

Bitmaps for *income_level*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Índices Bitmaps

- Los índices bitmaps son generalmente muy pequeños comparados con el tamaño de la relación.
  - Si los registros tiene 100 bytes, el tamaño de un array es 1/800 del tamaño de la tabla.
  - Si el número de valores distintos del bitmap es 8 entonces la relación de tamaño entre el índice y la tabla es 1/100.
- Los bitmaps son empaquetados en palabras del procesador, haciendo que su procesamiento sea muy rápido.
  - Bitmaps de 1 millón de bits puede ser procesados en 31250 instrucciones del procesador.
- Conteo de registros puede ser realizado rápidamente con bitmaps de existencia.
  - El numero de registros es igual al números de 1’s.
- Bitmaps pueden ser utilizados en las hojas de los índices B+ para reemplazar a los cajones de punteros cuando el valor indexado aparece en una proporción mayor a 1/W del total de registros, siendo W el tamaño en bits de 1 palabra del procesador.