

# Capítulo 4

## Almacenamiento de tablas en archivos e índices



# Organización de los discos rígidos

- La superficie del plato de un disco se divide en **pistas circulares**.
  - Suele haber en los discos rígidos entre 50 mil y 100 mil pistas por plato.
- Cada pista se divide en **sectores**.
  - Un sector es la unidad de datos más chica que puede ser leída o escrita.
  - Una pista suele tener entre 500 y 1000 sectores para pistas internas y entre 1000 y 2000 sectores para pistas externas.
  - El tamaño de un sector suele ser típicamente 512 B.
- Un **bloque** es una secuencia contigua de sectores de una pista.
  - Los datos son transferidos entre el disco y la memoria principal en bloques.
  - El tamaño de un bloque va desde 512 B a varios KiB.
  - El tamaño típico de un bloque va de 4 KiB a 16 KiB.

# Organización de archivos

- La BD es **almacenada** como una **colección de archivos**.
- Cada **archivo** es una secuencia de registros.
- Un **registro** es una secuencia de campos.
  - Pueden pensar una tabla de la BD como un conjunto de registros.
- Un enfoque:
  - El tamaño de **registro** es fijo
  - Cada **archivo** tiene registros de un tipo particular solamente
  - Diferentes **archivos** son usados para tablas diferentes de la BD.
- Este es el caso más fácil de implementar.

basicamente los valores o entradas dentro de una tabla

# Enfoque de registros de tamaño fijo

- **Idea:** Almacenar registro  $i$  desde el byte  $n \cdot (i-1)$ , donde  $n$  es el tamaño de cada registro.
  - Con esta idea registros pueden cruzar bloques.
- **Modificación:** que cada registro esté en un bloque.
- Analizamos el borrado de registros.
  - Vemos distintas alternativas para hacerlo.

# Borrar registro 3 y compactar

Borrar registro i:

- Mover registros  $i+1, \dots, n$  a  $i, \dots, n-1$ .

Si tenemos registros

1, 2, 3, 4, 5

Borrar 3 sería mover 4, 5 (todo lo mayor a 3)  
a la posición de 3

Es decir

1, 2, 3, 4, 5

<-----

1, 2, 4, 5, \_

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

# Borrar registro 3 y mover el último registro

Borrar registro i:

- Mover último registro n a i.

Es un SWAP de listas de TODA la vida  
la diferencia es que el campo i se  
elimina

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

header guarda direccion del primer borrado  
 el primer borrado guarda la direccion del  
 siguiente borrado  
 el segundo borrado guarda la direccion del  
 siguiente borrado  
 ...  
 ...  
 ...  
 Cuando no hay mas borrads el ultimo no guarda  
 puntero a nada

# Uso de listas enlazadas

No se mueven registros, pero se enlazan los registros libres en una lista.

- Almacenar la dirección del primer registro borrado en un encabezado del archivo.

header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

- Usar un registro borrado para almacenar la dirección del siguiente (en el tiempo) registro borrado.
- Estas direcciones son punteros, pues apuntan a localizaciones de registros.

# Organización de registros en archivos

- Con respecto a **la ubicación de las tablas** de la BD tenemos como alternativas:
  1. Cada tabla se aloja en archivo separado.
  2. Registros de diferentes tablas se alojan en el mismo archivo (se lo llama **agrupamiento de tablas en archivos**).
- Con respecto a **la ubicación de los registros** en un archivo:
  1. **Heap**: un registro puede almacenarse en cualquier lugar del archivo donde hay espacio.
  2. **Secuencial**: almacenar registros de una tabla en orden secuencial con respecto al valor de una clave de búsqueda de cada registro.

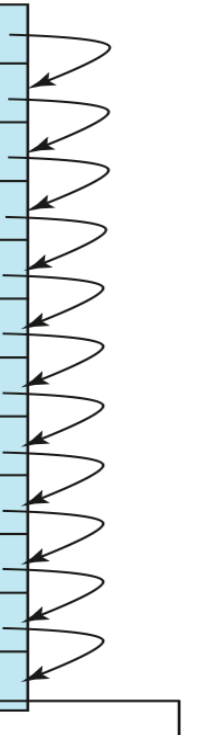
SQL  
SQL



# Organización de archivos secuencial

- Es adecuada para aplicaciones donde se requiere el procesamiento secuencial del archivo entero.
- Los registros en el archivo están **ordenados por clave de búsqueda**.
- Ejemplo: En la tabla siguiente la primera columna representa la clave de búsqueda.

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	



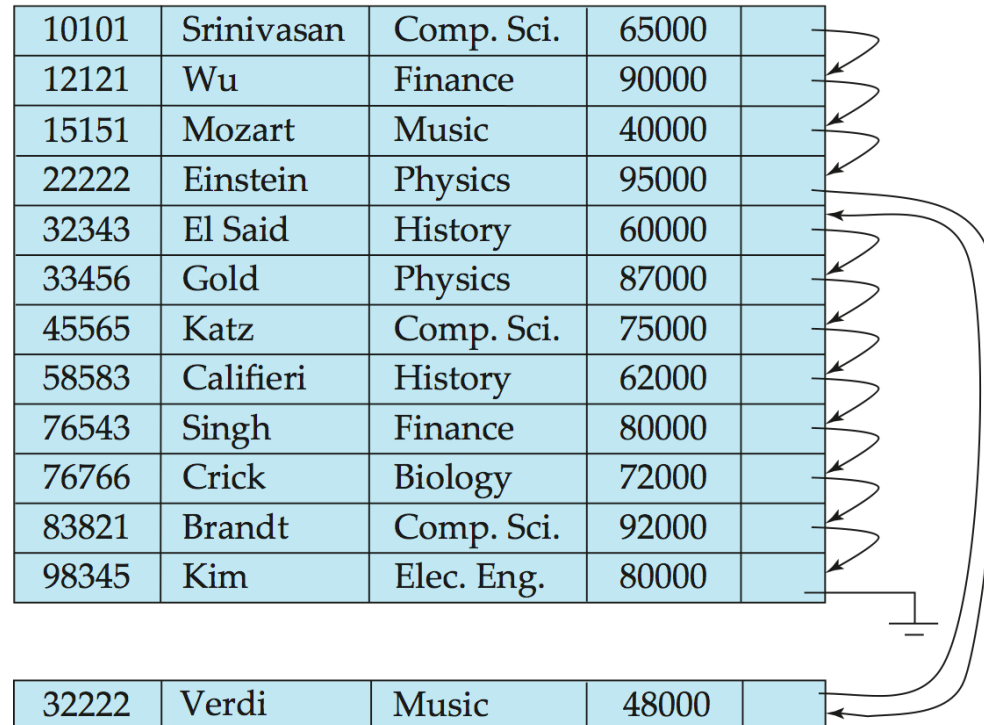
# Organización de archivos secuencial

- Algunas aclaraciones:

- Una **clave de búsqueda** es cualquier atributo o conjunto de atributos. No necesita ser la clave primaria o una superclave.
- Para permitir retorno rápido de registros en orden de clave de búsqueda encadenamos los registros por punteros.
- Un puntero en cada registro apunta al siguiente registro en el orden de la clave de búsqueda.
- Para minimizar el número de acceso a bloques en procesamiento de archivos secuenciales, almacenamos los registros físicamente en orden de clave de búsqueda o tan cercano al orden de clave de búsqueda como sea posible.

# Organización de archivos secuencial

- **Borrado:** usar cadenas de punteros.
- **Inserción:** localizar la posición donde el registro va a ser insertado.
  - Si hay espacio libre insertar allí.
  - Si no hay espacio libre, insertar el registro en **bloque de overflow**
  - En ambos casos el puntero de la cadena debe ser actualizado.
  - Hace falta reorganizar el archivo cada cierto tiempo para restaurar el orden secuencial.



# Organización de archivos secuencial

- **Evaluación:**

- A veces las consultas necesitan acceder a registros relacionados de dos tablas.
- Por ejemplo, calcular una reunión natural de dos tablas.
- Como veremos mas adelante, esto va a ser pesado de hacer si solo se usa organización de archivos secuencial.

# Agrupamiento de tablas en archivos

- Varias tablas van en un archivo.

*department*

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Comp. Sci.	Taylor	100000
Physics	Watson	70000

*instructor*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

Agrupamiento de *instructor* y *departamento* en un archivo.

- Luego de cada departamento se almacenan registros de instructores de ese departamento.

Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

# Agrupamiento de tablas en archivos

- **Evaluación:**

- Bueno para consultas involucrando *department* ⋈ *instructor* y para consultas involucrando un único departamento y sus instructores. Por ejemplo:

```
select dept name, building, budget, ID, name, salary
from department natural join instructor;
```

- Malo para consultas involucrando solo un departamento.
- Resulta en registros de tamaño variable.
- Se pueden agregar cadenas de punteros para enlazar registros de una relación particular.

Comp. Sci.	Taylor	100000	
45564	Katz	75000	
10101	Srinivasan	65000	
83821	Brandt	92000	
Physics	Watson	70000	
33456	Gold	87000	

# Acceso al almacenamiento

- Un archivo de la BD es particionado en unidades de almacenamiento de tamaño fijo llamados **bloques**.
- Los bloques son unidades tanto de alojamiento de almacenamiento como de transferencia de datos.
- El sistema gestor de BD busca minimizar la cantidad de transferencia de bloques entre disco y memoria.
  - Podemos reducir la cantidad de accesos a disco manteniendo tantos bloques como sea posible en memoria principal.
- **Búfer**: porción de memoria principal disponible para almacenar copias de bloques de disco.
- **Gestor de búfer**: subsistema responsable para alojar espacio de búfer en memoria principal.
  - Los programas llaman al gestor de búfer cuando necesitan un bloque del disco.

# Índices

- **Mecanismos de indexado** son usados para acceder más rápido a los datos deseados.
- **Clave de búsqueda** es un atributo o conjunto de atributos usados para buscar registros en un archivo.
- Un **archivo de índice** consiste de registros llamados **entradas del índice** de la forma:
  - <clave de búsqueda, puntero>
- Los archivos de índice son típicamente mucho más chicos que el archivo original.

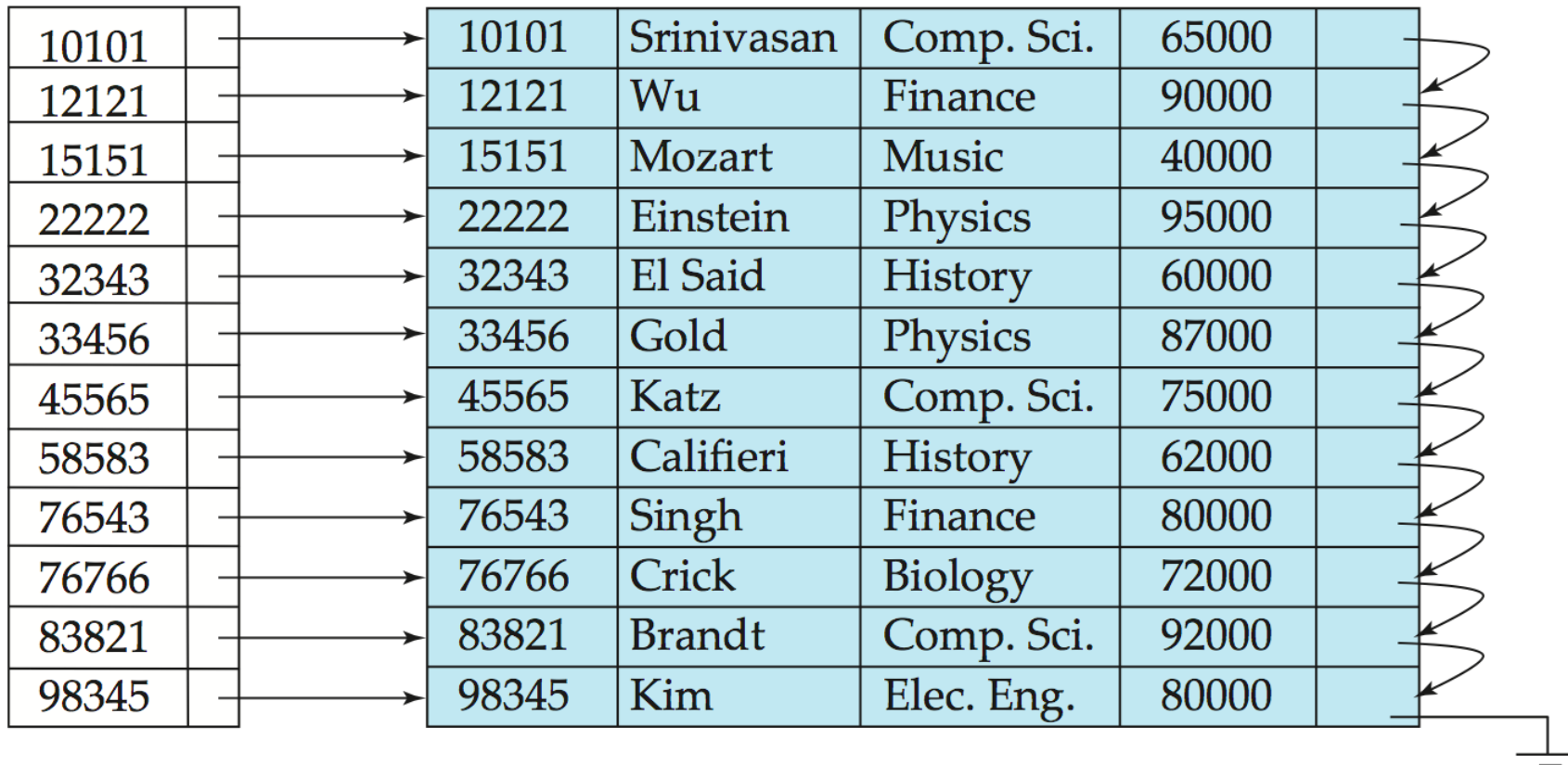


# Índices

- En un **índice ordenado** las **entradas del índice** están **almacenadas ordenadas por valor de la clave de búsqueda**.
- **Índice primario:**
  - Sea  $f$  un archivo secuencialmente ordenado;
  - el **índice primario** es el **índice cuya clave de búsqueda especifica el orden secuencial del archivo  $f$** .
- **Índice secundario:**
  - Sea  $f$  un archivo secuencialmente ordenado;
  - un **índice secundario** es un **índice cuya clave de búsqueda especifica un orden diferente del orden secuencial del archivo**.

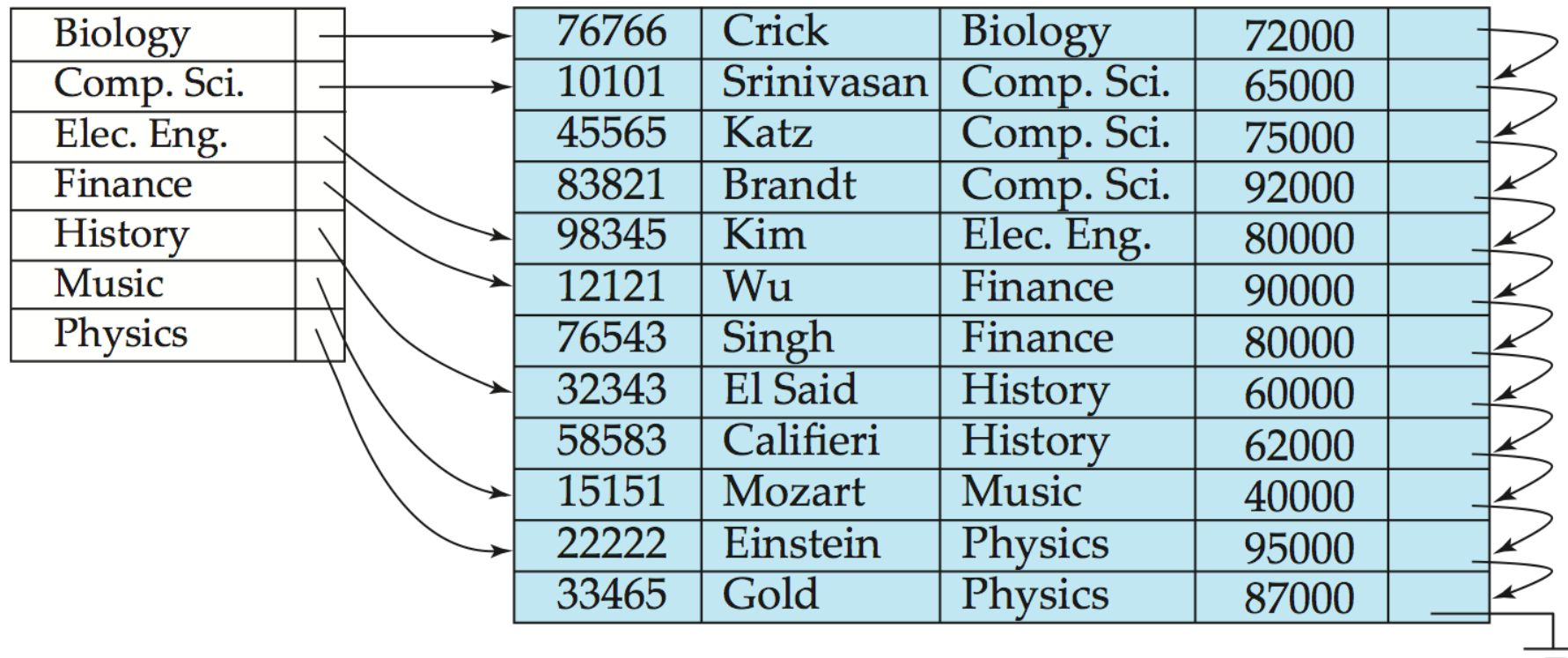
# Índice denso

- En un **índice denso** hay una entrada del índice por cada valor de la clave de búsqueda en el archivo.
- Índice en atributo ID de tabla de instructor (ID clave primaria).



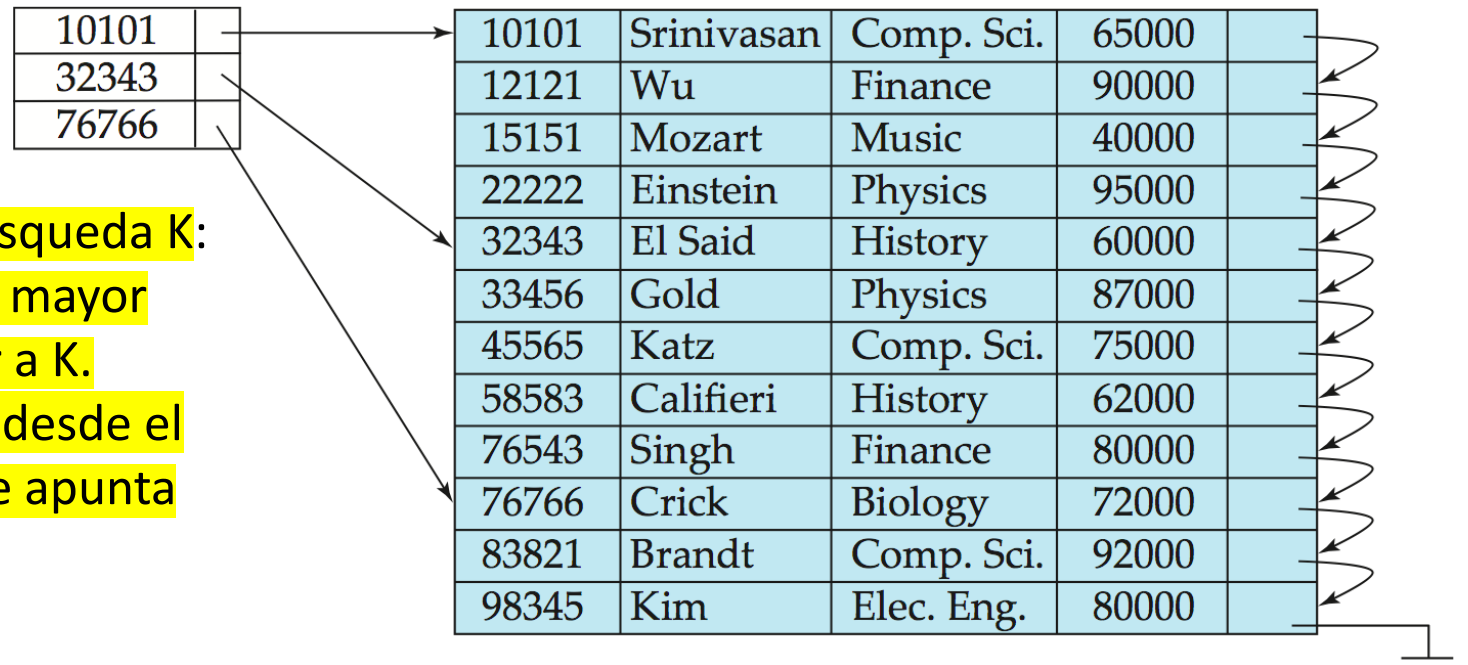
# Índice denso

- Índice denso en *nombre de departamento* con archivo de *instructor* ordenado por *nombre de departamento* (no clave primaria).
- Observar que los punteros en el índice apuntan al primer registro con ese *nombre de departamento* y siguiendo punteros en archivo de tabla se puede recorrer todos los registros de ese *nombre de departamento*.



# Índice disperso

- Un **índice disperso** contiene registros de **índice** para solo algunos valores de clave de búsqueda.
  - Se usan cuando los registros de la tabla están **secuencialmente ordenados por clave de búsqueda**.



Para ubicar un registro de clave de búsqueda K:

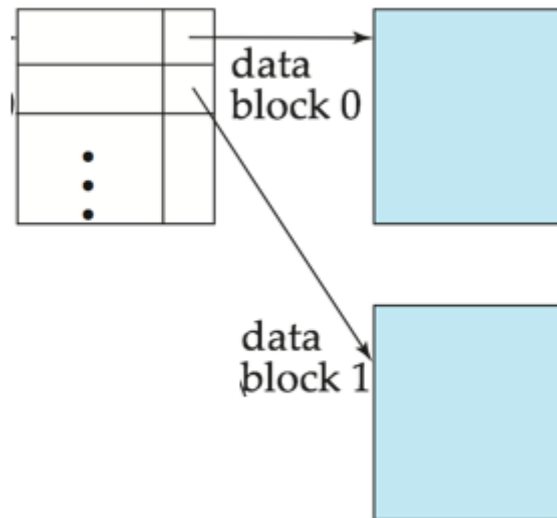
- Encontrar el registro del índice con mayor valor de clave de búsqueda menor a K.
- Buscar el archivo secuencialmente desde el registro al cual el registro del índice apunta

# Índice disperso

- **Evaluación:**

- Comparado con índices densos requieren menos espacio y menos sobrecarga de mantenimiento para inserciones y borrados en la tabla.
- Generalmente son más lentos que los índices densos para localizar registros.

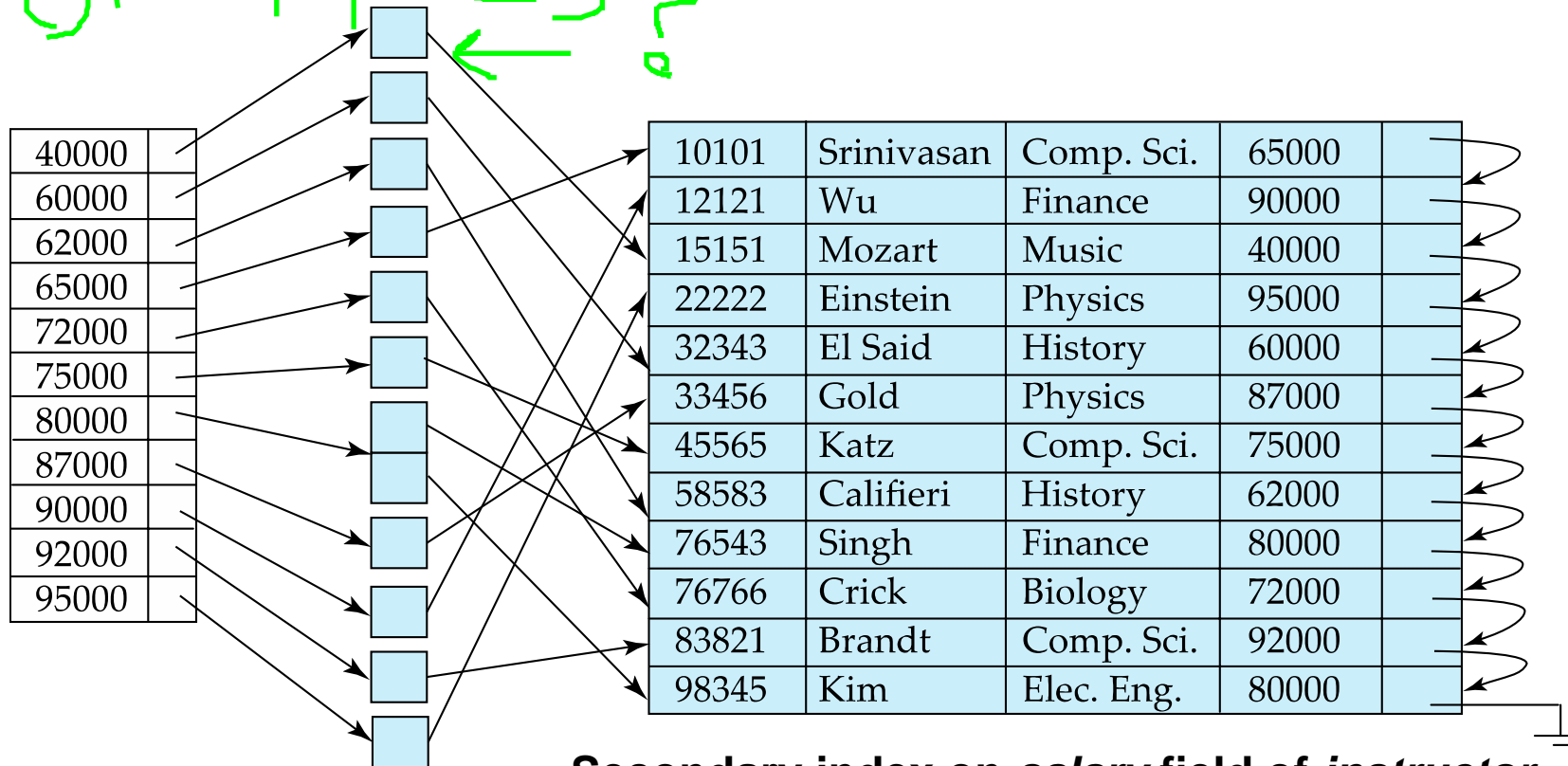
- **Ejemplo:** usar índice disperso con una entrada de índice para cada bloque del archivo de la tabla, correspondiente a la menor clave de búsqueda en el bloque.



# Índices secundarios

- Cada registro del índice apunta a un bucket que contiene punteros a todos los registros actuales con el valor particular de clave de búsqueda.
- Los índices secundarios deben ser densos.

SON FIJOS?



# Índices secundarios

- **Evaluación:**

- Escaneo secuencial usando índice primario es eficiente.
- Escaneo secuencial usando índice secundario es costoso.
  - Cada acceso a registro de tabla puede requerir recoger un nuevo bloque de disco.
  - Recoger un bloque requiere entre 5 y 10 msec, versus 100 nanosecs para acceso a memoria.

# Índices

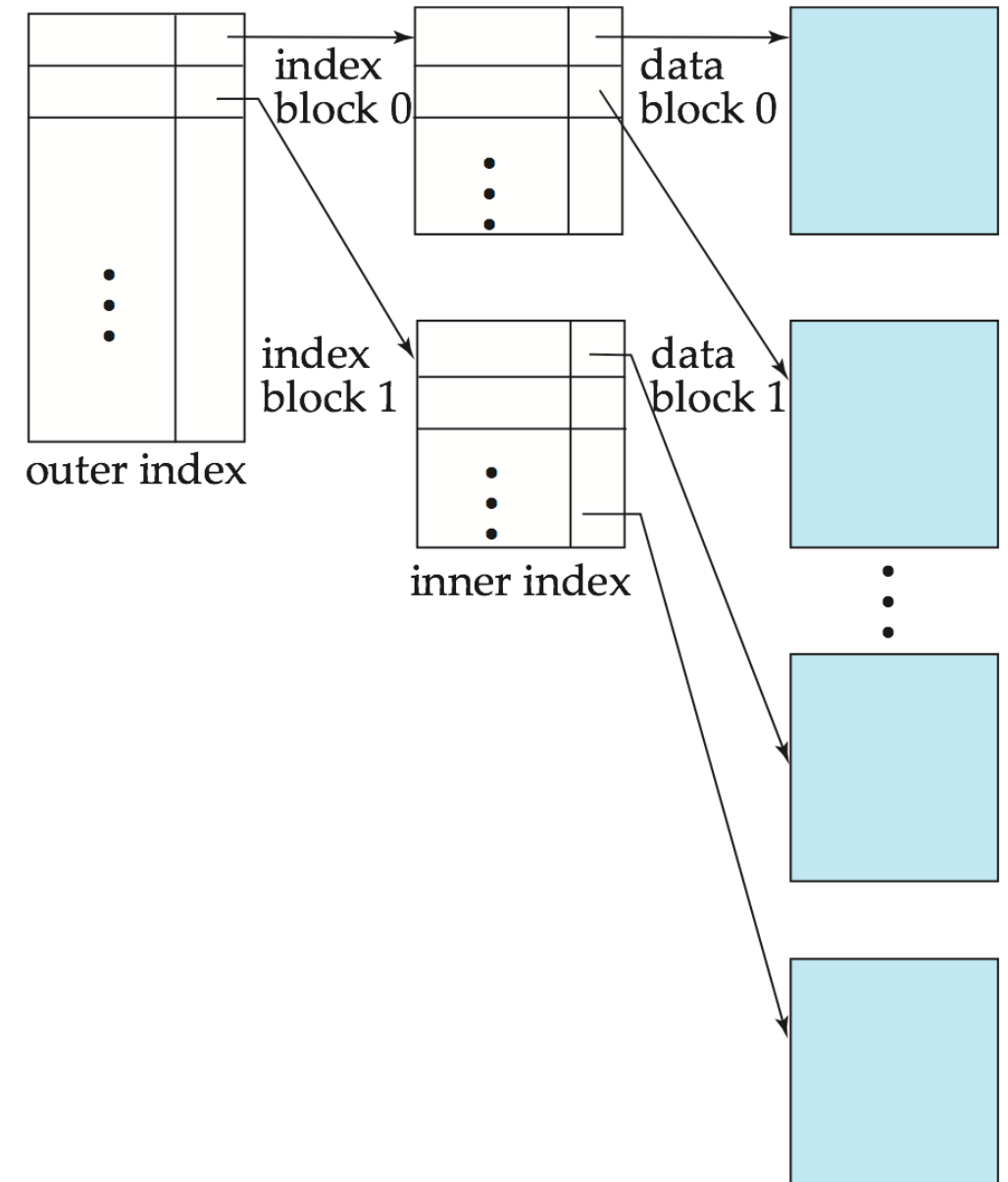
- **Evaluación (del uso de índices):**

- Índices ofrecen beneficios sustanciales cuando se accede a registros de tablas.
- Actualizar índices impone una sobrecarga en la modificación de la BD.
  - Cuando un archivo se modifica, cada índice al archivo debe ser actualizado.



# Índice de multinivel

- **Problema:** Si el índice primario no cabe en memoria, entonces el acceso pasa a ser costoso.
- **Solución:** tratar el índice primario mantenido en disco como archivo secuencial y construir un índice disperso en el mismo.
  - **Índice externo:** índice disperso del índice primario.
  - **Índice interno:** archivo del índice primario.
- **Evaluación**
  - Índices de los dos niveles deben ser actualizados al insertar y borrar en la tabla.



# Actualización de índice por borrado de registro

- Índice de un solo nivel

- Índice denso:

- Si el registro borrado fue el único registro en el archivo con su clave de búsqueda particular: la clave de búsqueda es borrada del índice también.

- sino:

- Si la entrada del índice almacena punteros a todos los registros con el mismo valor de clave de búsqueda: se borra el puntero al registro borrado de la entrada del índice.
    - Sino: si el registro borrado es el primer registro con el valor de la clave de búsqueda, se actualiza la entrada del índice para que apunte al siguiente registro.

# Actualización de índice por borrado de registro

- **Índice de un solo nivel (Cont)**

- **Índice disperso:**

- Si el índice no contiene una entrada con el valor de **clave de búsqueda del registro borrado**: nada necesita hacerse en el índice.
    - **Sino:**
      - Si el registro borrado era el único registro con su **clave de búsqueda B**:
        - Si el **próximo valor de clave de búsqueda tiene una entrada en el índice**: la entrada para B es borrada del índice.
        - **Sino**: se reemplaza el registro de índice correspondiente con un registro de índice para el próximo valor de clave de búsqueda (en el orden de clave de búsqueda).
      - **Sino: si la entrada del índice para la clave de búsqueda apunta al registro siendo borrado**: se actualiza la entrada del índice para apuntar al próximo registro con el mismo valor de clave de búsqueda.

# Actualización de índice por inserción de registro

- **Índice de un solo nivel:**

- Primero se hace búsqueda usando el valor de clave de búsqueda que aparece en el registro a ser insertado.
- **Índice denso:**
  - **Si el valor de la clave de búsqueda no aparece en el índice:** insertar una entrada de índice con valor de clave de búsqueda en el índice en la posición apropiada.
  - **Sino:**
    - **Si la entrada del índice almacena punteros a todos los registros con el mismo valor de clave de búsqueda:** el sistema agrega un puntero al nuevo registro en la entrada del índice.
    - **Sino:** la entrada del índice almacena un puntero a solo el primer registro con el valor de la clave de búsqueda. Se coloca el registro siendo insertado luego de los otros registros con los mismos valores de clave de búsqueda.

# Actualización de índice por inserción de registro

- **Índice de un solo nivel (Cont.):**

- **Índice disperso:**

- Asumimos que el índice almacena una entrada para cada bloque.
    - **Si el sistema crea un nuevo bloque:** inserta el primer valor de la clave de búsqueda (en orden de clave de búsqueda) apareciendo en el nuevo bloque en el índice.
    - **sino:**
      - **Si el nuevo registro tiene el menor valor de clave de búsqueda en su bloque:** el sistema actualiza la entrada del índice apuntando al bloque.
      - **Sino:** el sistema no hace cambios al índice.

# Actualización de índice de varios niveles

- Algoritmos de inserción y borrado para índices de varios niveles son una extensión simple del esquema ya descrito.
- **Luego de inserción o borrado de registro** el sistema actualiza el índice de más bajo nivel como se describió.
- Con respecto al segundo nivel, el índice de más bajo nivel es meramente un archivo conteniendo registros.
- **Si hay algún cambio en el índice de más bajo nivel:** el sistema actualiza el índice de segundo nivel como se describió.

# Índices con varias claves

- Una clave de búsqueda conteniendo más de un atributo se llama **clave de búsqueda compuesta**.
- La estructura del índice es la misma como la de cualquier otro índice; la única diferencia es que la clave de búsqueda del índice contiene una lista de atributos.
- La clave de búsqueda puede representarse como una tupla de valores de la forma:  $(a_1, \dots, a_n)$ , donde los atributos individuales son:  $A_1, \dots, A_n$ .
- El orden de los valores de la clave de búsqueda es el **orden lexicográfico**.
- P. ej: si la clave de búsqueda tiene 2 atributos:  $(a_1, a_2) < (b_1, b_2)$  si  $a_1 < b_1$ , o  $a_1 = b_1$  y  $a_2 < b_2$ .

# Acceso a múltiples claves de búsqueda

- Usar múltiples índices para ciertos tipos de consultas.
- Ejemplo:  
    **select** ID  
    **from** instructor  
    **where** dept\_name = 'Finance' and salary = 80000
- Posibles estrategias para procesar consultas usando índices en atributos simples:
  - ☐ Usar índice en *dept\_name* para encontrar instructores con nombre de departamento 'Finance'; testear *salary* = 80000.
  - ☐ Usar índice en *salary* para encontrar instructores con *salary* de 80000; testear *dept\_name* = 'Finance'.
  - ☐ Usar índice en *dept\_name* para encontrar punteros a todos los registros pertenecientes al departamento de finanzas. Similarmente usar índice en salario para hallar punteros a registros con salario de 80000. Tomar la intersección de ambos conjuntos de punteros obtenidos.



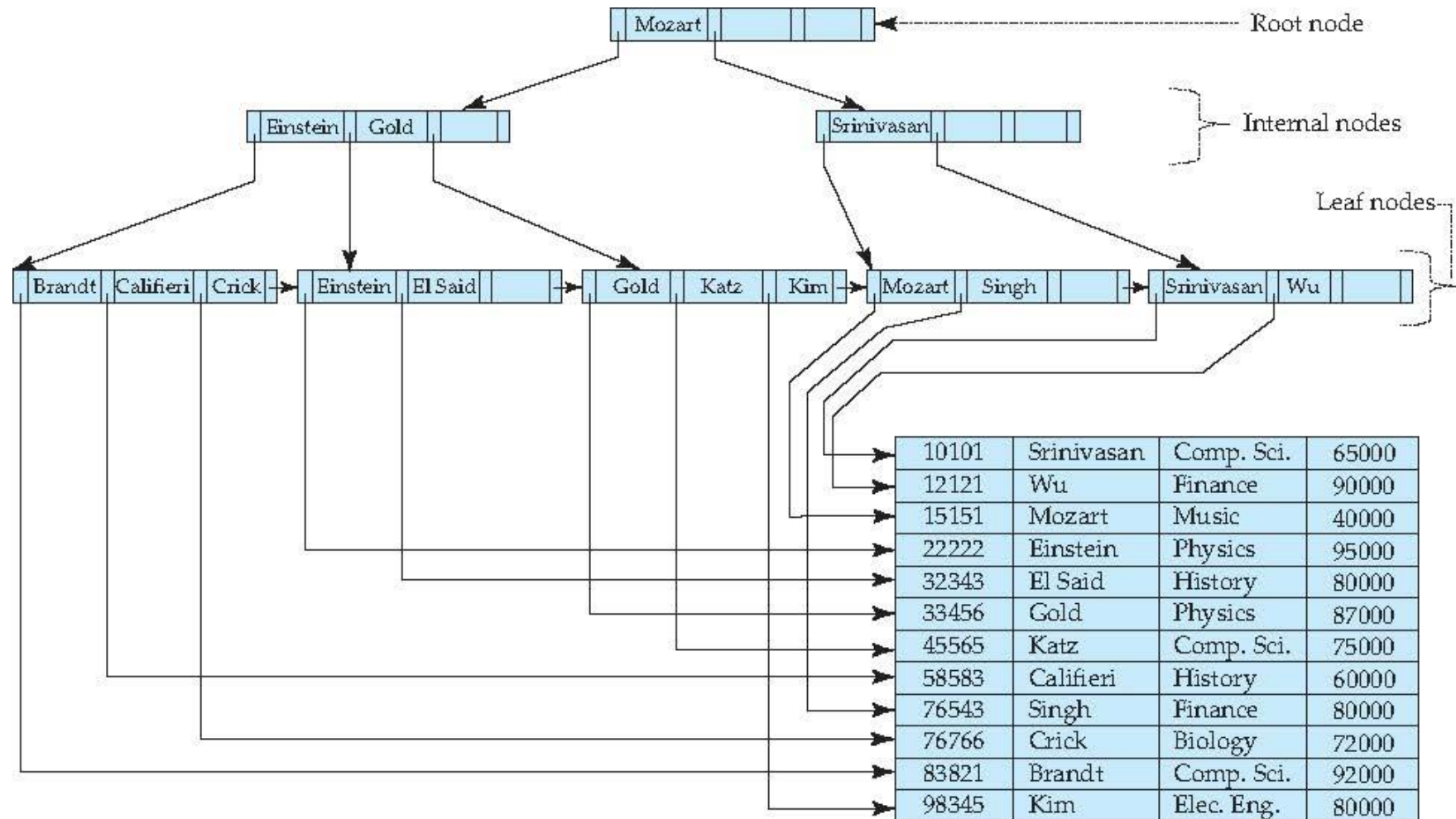
# Índices en varios atributos

- Supongamos que tenemos un índice en la clave de búsqueda combinada (*dept\_name*, *salary*).
- Con la clausula **where** anterior el índice en (*dept\_name*, *salary*) puede ser usado para recoger solo registros que satisfacen ambas condiciones.
- Además se puede manejar eficientemente:  
**where** dept\_name = 'Finance' and salary < 80000
- Pero no se puede manejar eficientemente:  
**where** dept\_name < 'Finance' and salary = 80000
  - Puede recolectar varios registros que satisfacen la primera, pero no la segunda condición.

# Índices árbol B<sup>+</sup>

- **Índices árbol B<sup>+</sup>** son una alternativa para el indexado de archivos secuenciales.
- **Desventaja de archivos secuenciales indexados:**
  - El desempeño se degrada a medida que el archivo crece, porque muchos bloques overflow son creados.
  - La reorganización periódica del archivo entero es requerida para remediar esa degradación.
- Un **índice de árbol B<sup>+</sup>** toma la forma de un **árbol balanceado** en el cual todo camino de la raíz a una hoja tiene la misma longitud.
- **Ventaja de archivos índice árbol B<sup>+</sup>:**
  - Se reorganizan a si mismos con cambios pequeños locales frente a inserciones o borrados.
  - La reorganización del archivo entero no es requerida para mantener el desempeño.
- **Desventaja menor de árboles B<sup>+</sup>:** sobrecarga extra de inserción y borrado, sobrecarga de espacio. Pero las mismas son aceptables.
- Las ventajas de árboles B<sup>+</sup> superan a las desventajas. Los árboles B<sup>+</sup> son usados extensivamente.

# Ejemplo de árbol B<sup>+</sup>

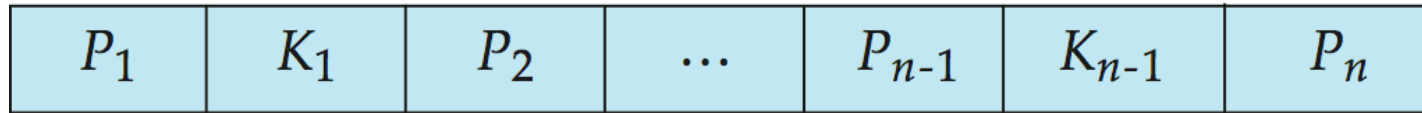


# Índices árbol B<sup>+</sup>

- **Un árbol B<sup>+</sup> satisface las siguientes propiedades:**
  - Todos los caminos desde la raíz a una hoja son de la misma longitud.
  - Cada nodo que no es la raíz o una hoja tiene entre  $\lceil n/2 \rceil$  y  $n$  hijos.
  - Un nodo hoja tiene entre  $\lceil (n-1)/2 \rceil$  y  $n-1$  valores.
  - Si la raíz no es una hoja, tiene al menos 2 hijos.
  - Si la raíz es una hoja, puede tener entre 0 y  $n-1$  valores.

# Estructura de nodo de árbol B<sup>+</sup>

- **Nodo típico:**

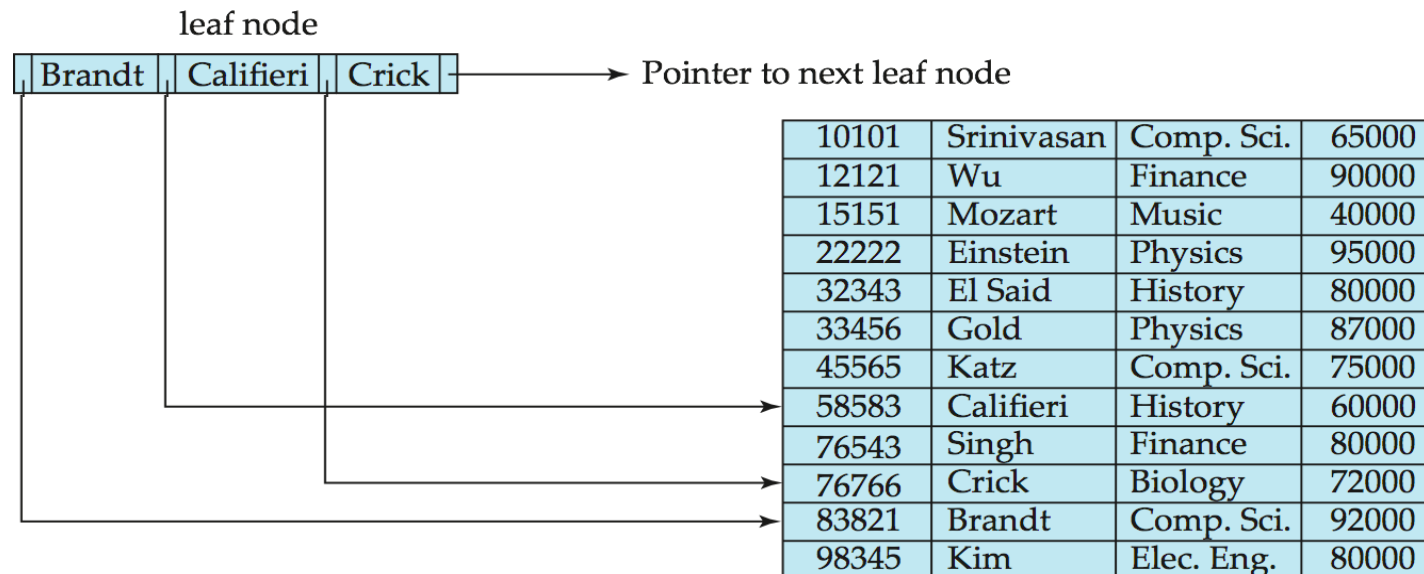


- Los  $K_i$  son valores de clave de búsqueda.
- Los  $P_i$  son:
  - Para **nodos no hoja**: punteros a hijos
  - Para **nodos hoja**: punteros a registros o buckets de registros.
- Las claves de búsqueda en un nodo están ordenadas.
$$K_1 < K_2 < K_3 < \dots < K_{n-1}$$
- Los rangos de valores en cada hoja no se solapan, excepto cuando hay valores de clave de búsqueda duplicados, en cuyo caso un valor puede estar presente en más de una hoja.
- Asumir inicialmente que no hay claves duplicadas; manejar duplicaciones después.

# Nodos hoja de un árbol B<sup>+</sup>

- **Propiedades de un nodo hoja:**

- Para  $i = 1, 2, \dots, n-1$ , el puntero  $P_i$  apunta al registro del archivo con el valor de clave de búsqueda  $K_i$ .
- Si  $L_i, L_j$  son nodos hoja y  $i < j$ , los valores de la clave de búsqueda de  $L_i$  son menores o iguales a los valores de la clave de búsqueda de  $L_j$ .
- $P_n$  apunta al próximo nodo hoja en el orden de clave de búsqueda.

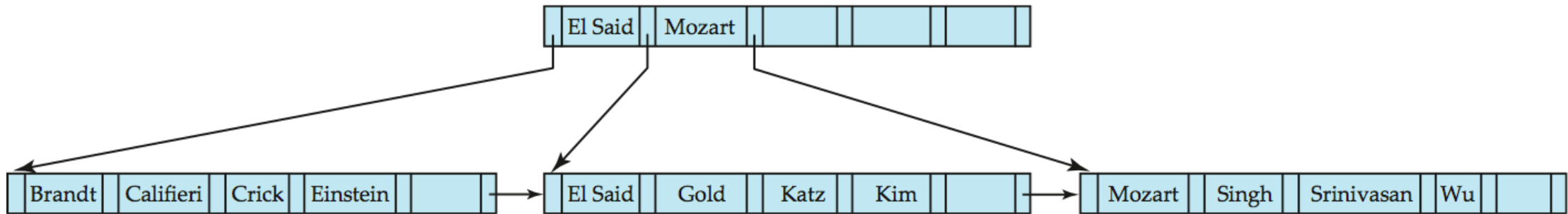


# Nodos no hoja en árboles B<sup>+</sup>

- Los **nodos no hoja** forman un índice disperso multinivel en los nodos hoja. Para un nodo no hoja con  $n$  punteros:
  - Todas las claves de búsqueda en el subárbol al cual  $P_1$  apunta son menores que  $K_1$ .
  - Para  $2 \leq i \leq n - 1$  : todas las claves de búsqueda en el subárbol al cual  $P_i$  apunta tienen valores mayores o iguales a  $K_{i-1}$  y menores que  $K_i$ .
  - Todas las claves de búsqueda en el subárbol al cual  $P_n$  apunta tiene valores mayores o iguales que  $K_{n-1}$ .

$P_1$	$K_1$	$P_2$	$\dots$	$P_{n-1}$	$K_{n-1}$	$P_n$
-------	-------	-------	---------	-----------	-----------	-------

# Ejemplo de árbol B<sup>+</sup>



Árbol B<sup>+</sup> para tabla *instructor* ( $n = 6$ )



# Observaciones

- Como las conexiones entre nodos son hechas mediante punteros, los bloques lógicamente cercanos no necesitan estar físicamente cercanos.
- Si hay  $K$  valores de clave de búsqueda en la tabla, la altura del árbol  $B^+$  no es mayor a  $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$ .
  - Entonces las consultas pueden ser realizadas eficientemente.
- Borrados e inserciones a la tabla pueden ser manejados eficientemente, porque el índice puede ser reestructurado en tiempo logarítmico.

# Consultas en árboles B<sup>+</sup>

- **Encontrar registro con valor de clave de búsqueda  $V$ .**
- *Function find(value  $V$ )*
  1.  $C = \text{root}$
  2. While  $C$  is not nodo hoja {
    1. Sea  $i$  el menor valor tal que  $V \leq K_i$ .
    2. If no existe: set  $C = \text{último puntero non-null en } C$
    3. Else { if ( $V = K_i$ ):  $C = P_{i+1}$  else:  $C = P_i$  }}
  3. Sea  $i$  el menor valor tal que  $K_i = V$
  4. If existe tal valor  $i$ : seguir el puntero  $P_i$  al registro deseado.
  5. Else no existe registro con valor de clave de búsqueda  $k$ .

# Manejando duplicados

- **Con claves de búsqueda duplicadas** (tanto en nodos hoja como internos):
  - No podemos garantizar que  $K_1 < K_2 < K_3 < \dots < K_{n-1}$
  - Pero podemos garantizar que  $K_1 \leq K_2 \leq K_3 \leq \dots \leq K_{n-1}$
- Las claves de búsqueda en el subárbol al cual  $P_i$  apunta
  - son  $\leq K_i$ , pero no necesariamente  $< K_i$ .
  - Para ver por qué, supongamos que el mismo valor de la clave de búsqueda  $V$  está presente en dos nodos hoja  $L_i$  y  $L_{i+1}$ .
    - Entonces en el nodo padre  $K_i$  debe ser igual a  $V$ .

# Manejando duplicados

- **Modificamos el procedimiento de búsqueda como sigue:**
  - Recorrer  $P_i$  *incluso* si  $V = K_i$
  - Tan pronto como alcanzamos un nodo hoja  $C$ , chequear si  $C$  tiene solo valores de clave de búsqueda menores que  $V$ .
    - Si es así, sea  $C =$  hermano derecho de  $C$  antes de chequear si  $C$  contiene  $V$ .
- **Procedimiento *printAll***
  - Usa búsqueda modificada para encontrar primera ocurrencia de  $V$ .
  - Recorre las hojas consecutivas para encontrar todas las ocurrencias de  $V$ .

# Consultas en árboles B<sup>+</sup>

- Si hay  $K$  valores de clave de búsqueda en el archivo, la altura del árbol no es mayor a  $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$ .
- Un nodo tiene generalmente el mismo tamaño que un bloque de disco, típicamente 4 KiB.
  - y  $n$  es típicamente cercano a 100 (40 B por entrada de índice).
- **Ejemplo:** Con 1 millón de valores de clave de búsqueda y  $n = 100$ 
  - A lo más  $\log_{50}(1,000,000) = 4$  nodos son accedidos en una búsqueda.

# Lectura recomendada

- Leer sobre otras operaciones en árboles árbol  $B^+$ :
  - Inserción, borrado.

# Indexando strings

- **Problema:** Creando índices árbol B<sup>+</sup> en atributos de valor string da lugar a dos problemas:
  - los string pueden ser de longitud variable y
  - los string pueden ser largos dando lugar a menor cantidad de punteros por nodo y llevando a aumentar la altura del árbol.
- **Notar que:**
  - Un nodo debe ser subdividido si está lleno sin importar cuantas entradas de búsqueda contenga.
  - Nodos pueden ser combinados o sus entradas redistribuidas dependiendo de qué fracción del espacio en los nodos es usado,
    - en lugar de basarse en el número máximo de entradas que un nodo puede contener.

# Indexando strings

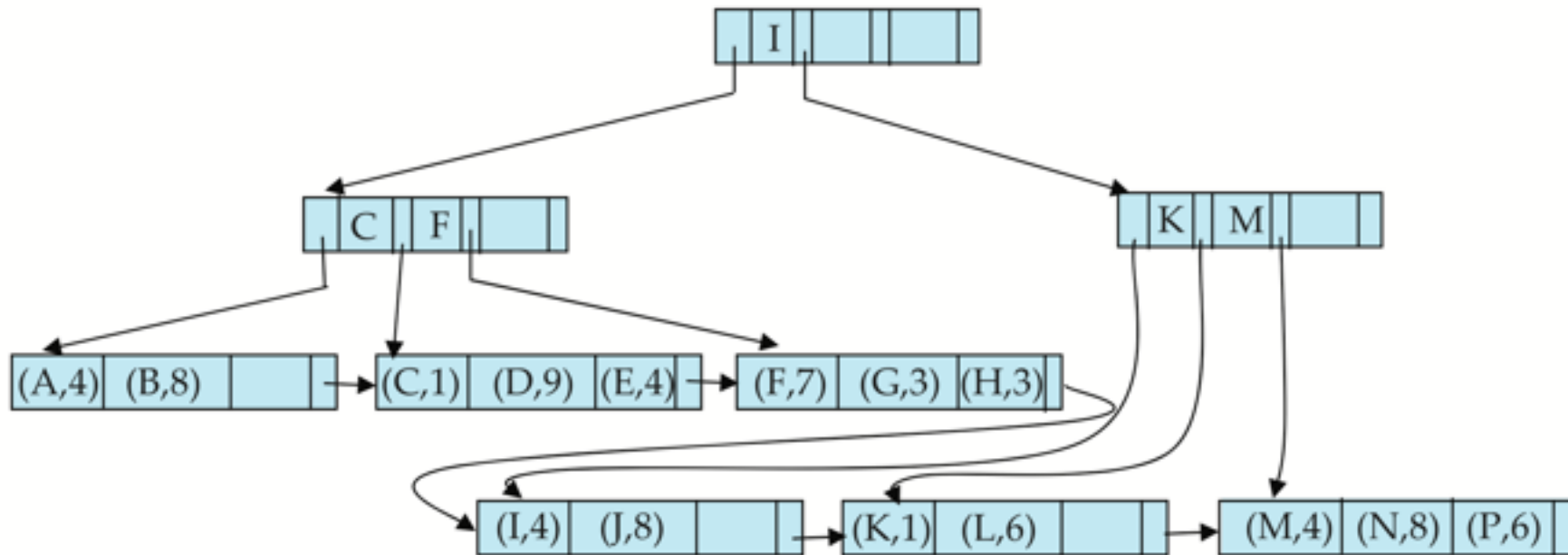
- **Solución:** La cantidad de punteros de un nodo puede ser aumentada usando una técnica llamada **compresión de prefijos**.
  - En los nodos no hoja no almacenamos el valor entero de la clave de búsqueda.
  - Solo almacenamos un **prefijo** de cada valor de clave de búsqueda que es suficiente para distinguir entre los valores de las clave de búsqueda en los subárboles que separa.
- **Ejemplo:** si tenemos un índice en nombres, el valor de la clave de búsqueda en un nodo no hoja puede ser un prefijo de un nombre.
  - Puede ser suficiente almacenar 'Silb' en un nodo no hoja en lugar de 'Silberschatz' si el valor más cercano en los dos subárboles que separa son 'Silas' y 'Silver' respectivamente.



# Organización de archivo con árbol B<sup>+</sup>

- El problema de la degradación de archivo de índice secuencial se resuelve usando índice de árbol B<sup>+</sup>
- El problema de la degradación de un archivo de datos se resuelve usando una organización de archivo de árbol B<sup>+</sup>
- Los nodos hoja en una organización de archivo de árbol B<sup>+</sup> almacena registros en lugar de punteros.
- Los nodos hoja son todavía requeridos que estén la mitad llenos.
  - Como los registros ocupan más espacio que los punteros, el máximo número de registros que pueden ser almacenados en un nodo hoja es menor que el número de punteros en un nodo no hoja.
- La inserción y borrado so manejados de la misma manera que inserción y borrado de entradas en índices de árbol B<sup>+</sup>

# Organización de archivo con árbol B<sup>+</sup>



Ejemplo de organización de archivo con árbol B<sup>+</sup>