Dpto. Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Elementos de Bases de Datos

Segundo Cuatrimestre 2015

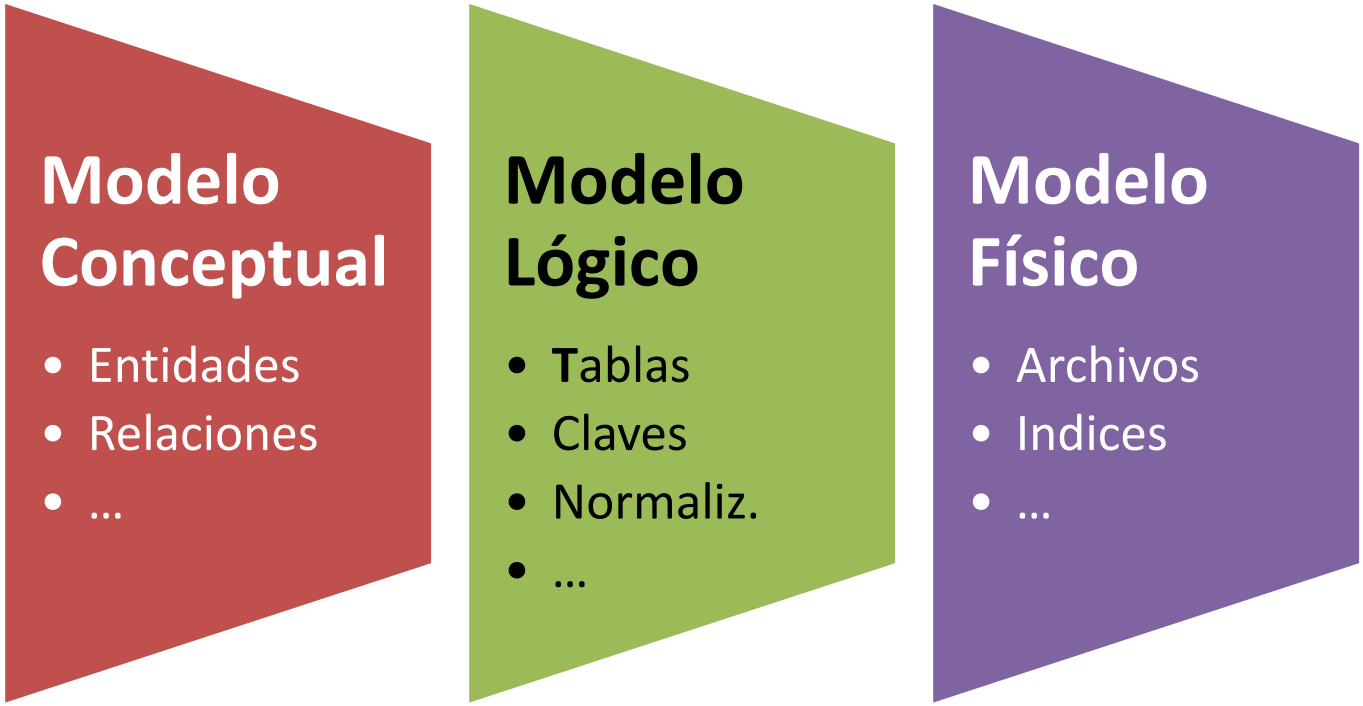
Clase 11:

Nivel Físico (Parte I) – Almacenamientos e Índices

Mg. María Mercedes Vitturini
[mvitturi@uns.edu.ar]

Modelos de Datos - Abstracciones

- Tradicionalmente, en el área de bases de datos, en el **diseño del modelo de datos de una aplicación** se distinguen según el nivel de abstracción los modelos: *conceptual, lógico y físico*.



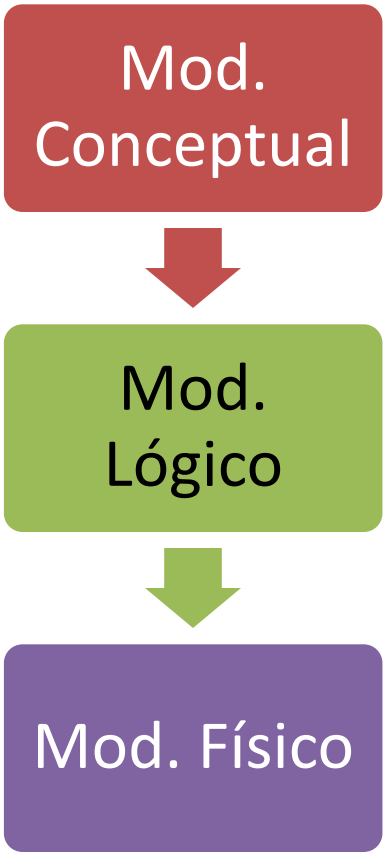
Almacenamiento Conceptos

- Nivel Físico – El modelo físico
- Clasificación
  - Almacenamiento Primario → Volatil
  - Almacenamiento Secundario → No Volatil – On line
  - Almacenamiento Terciario → No Volatil – Off line
- Medidas de calidad
- Disco Magnético
  - Acceso
  - Evaluación
  - Stripping y Mirroring
  - RAIDs - Alternativas

Nivel Físico

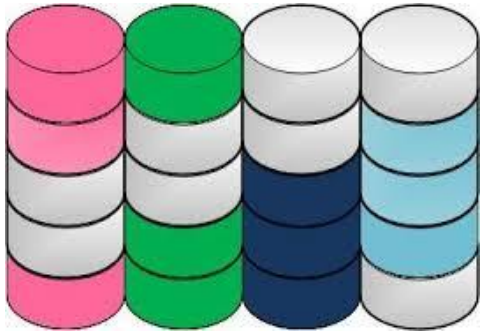
El **Nivel Físico** determina cómo el DBMS representa y almacena los datos en el/los distintos medios de almacenamiento.

- El esquema físico describe la **organización física de los registros, la elección de organización de archivos, el uso de índices, etc.**
- El objetivo del diseño físico es favorecer el *manejo eficientemente de los datos (agregar, borrar, modificar, guardar y recuperar).* Interesa el equilibrio entre espacio, tiempo y confiabilidad.



Almacenamiento Físico

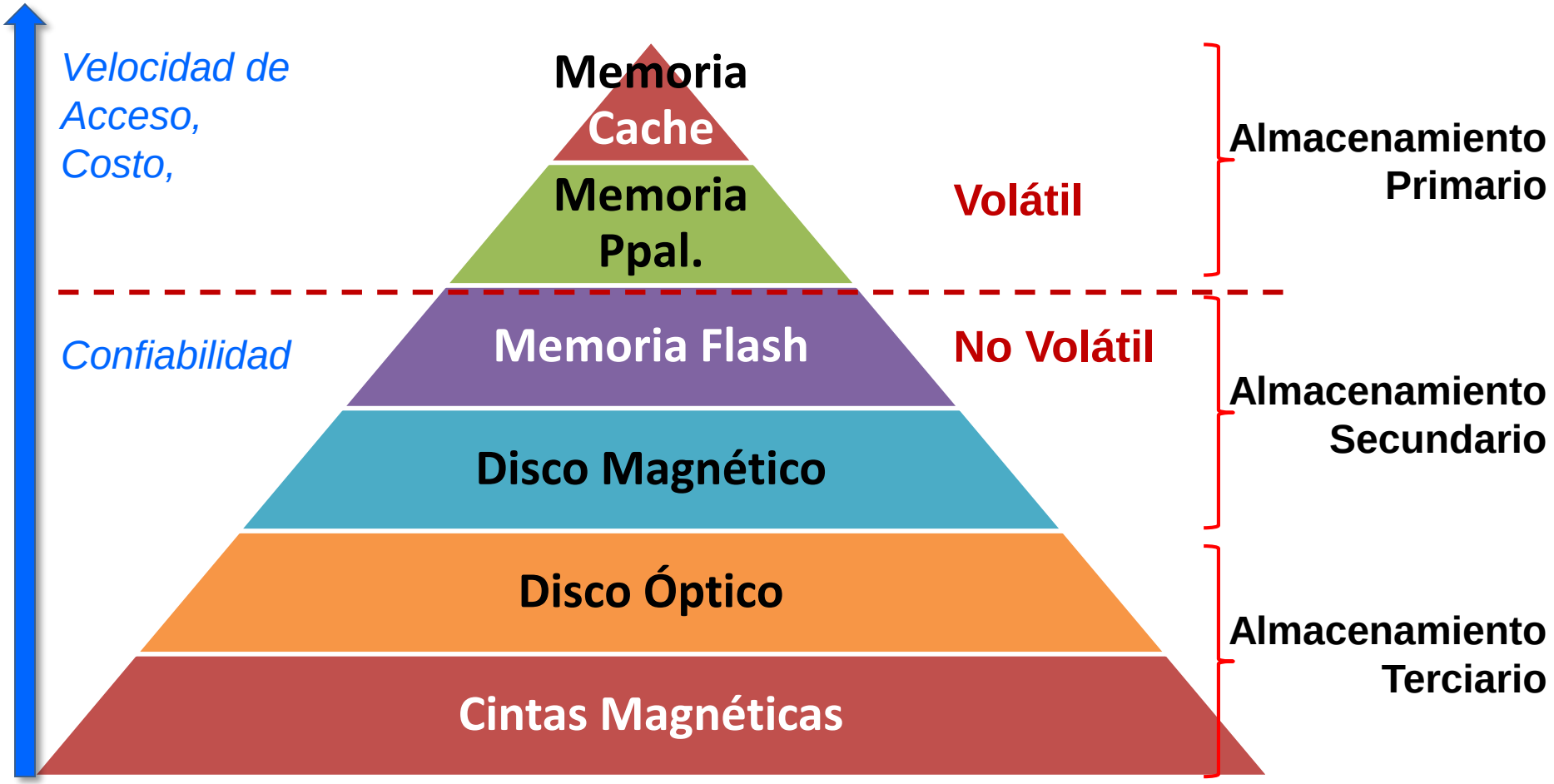
- Las características físicas de los distintos dispositivos de almacenamiento (*hardware*) determinan el modo en que se almacenan los datos en cuanto:
  - **Capacidad**
  - **Velocidad de Acceso**
  - **Costo por unidad**
  - **Confiabilidad**



- Recordamos

"Un SMBD es un sistema de computación responsable del *almacenamiento y recuperación eficiente y de los elementos persistentes de una BD*"

Jerarquía de Almacenamiento Físico



Medios de Almacenamiento Físico

MEMORIA CACHE: el medio de almacenamiento más rápido y más costoso. Se maneja por hardware. La capacidad es "limitada".

MEMORIA PRINCIPAL: medio que almacena los datos operativos (las instrucciones de máquina operan con datos en memoria). Las nuevas tecnologías incrementaron notablemente el tamaño de la memoria, igualmente es chica y costosa como para cargar la base de datos completa. No sobrevive a las caídas de sistema.

MEMORIA FLASH: De capacidad intermedia (pen-drives, discos de estado solido). Sobrevive a los cortes de sistema. La lectura es muy eficiente

DISCOS MAGNÉTICOS: medio primario para almacenar datos en el tiempo y disponibles en línea. DE acceso directo. El sistema mueve los datos desde/hacia disco a memoria para operar.

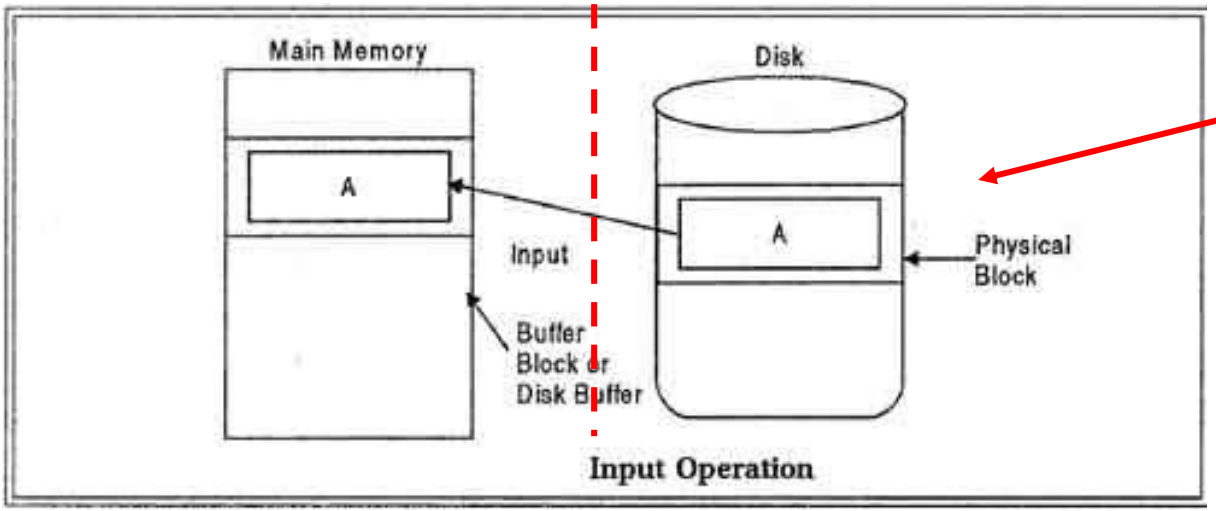
DISCOS ÓPTICOS: CD /DVD

CINTAS MAGNÉTICAS: para backup y archivo. El acceso es lento y secuencial.

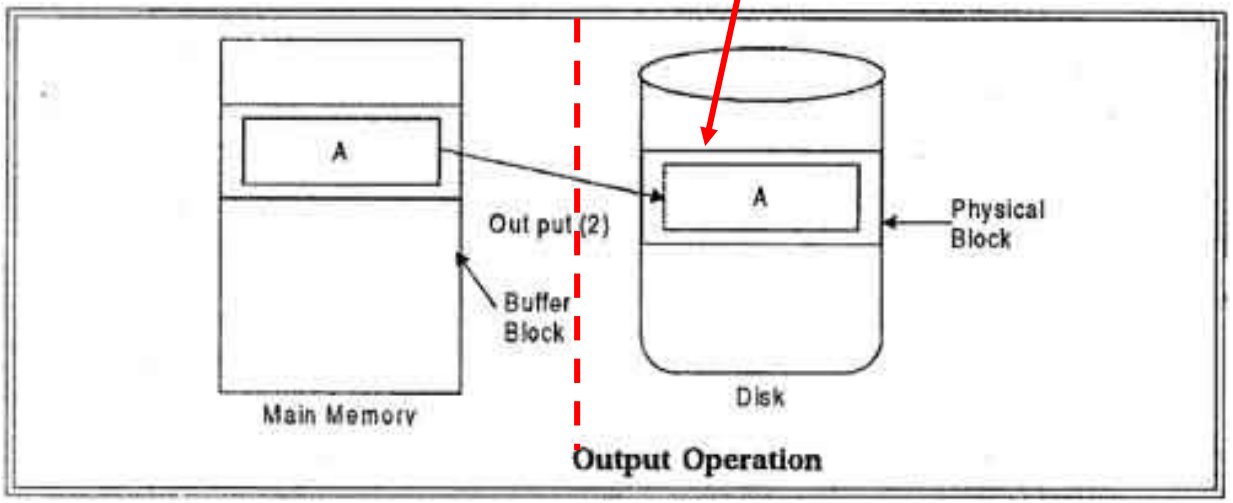
Medidas para discos magnéticos.

- **TIEMPO DE ACCESO:** tiempo transcurrido desde un pedido de lectura/escritura hasta que comienza la transferencia de datos.
  - *Tiempo de búsqueda*.
  - *Tiempo de latencia por rotación*.
- **TIEMPO DE TRANSFERENCIA:** velocidad a la que se recuperan o guardar datos.
  - Influye el controlador si maneja varios discos.
- **TIEMPO MEDIO ENTRE FALLOS:** tiempo promedio que se puede esperar que funcione sin fallos.

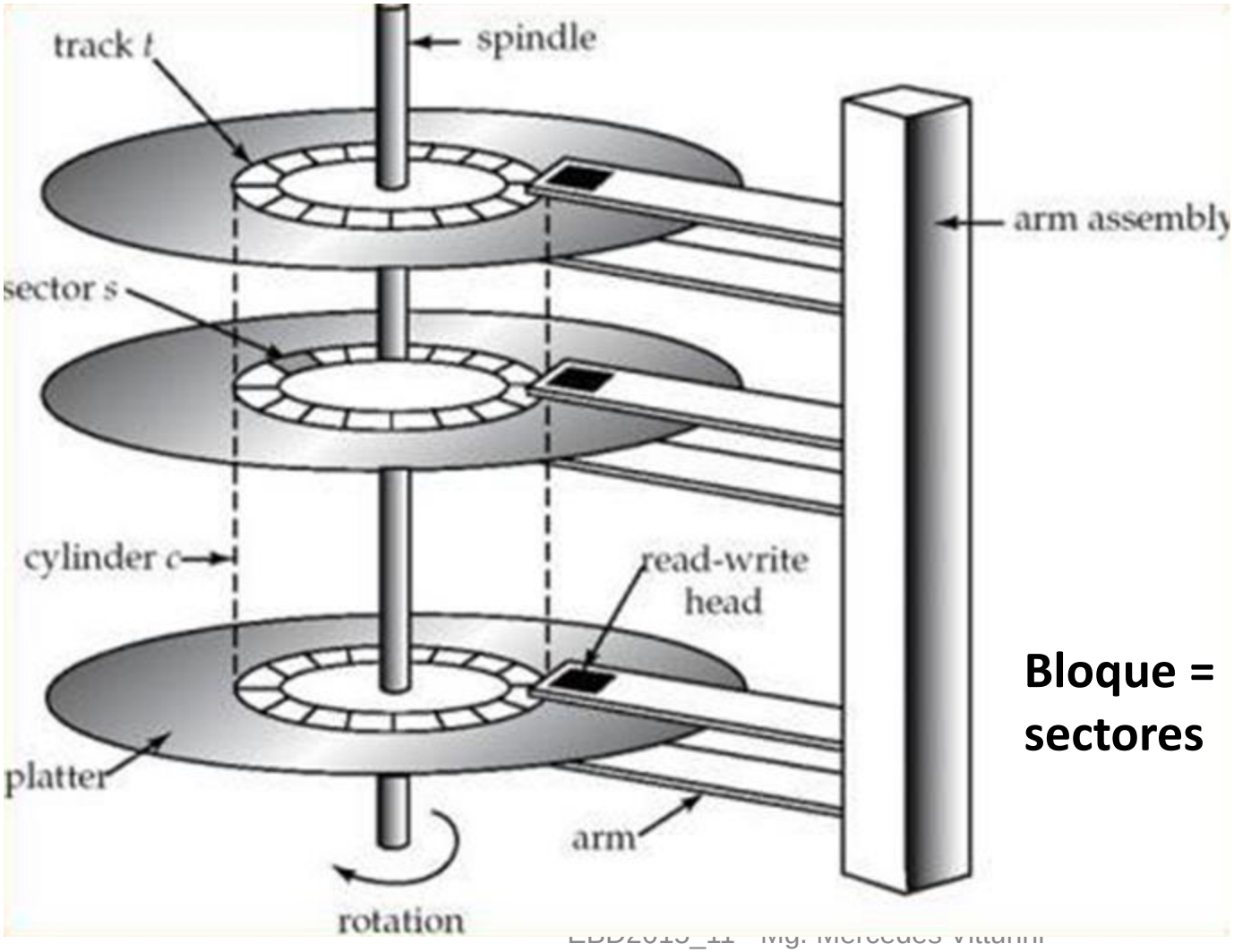
Acceso a Disco



Bloque o página es la unidad de transferencia



Disco Magnéticos - Estructura



Bloque = secuencia de sectores

Acceso a bloques de disco

- Una **solicitud de E/S** a disco especifica la dirección del disco en la forma de un *número de bloque*.
- Un **bloque** es una secuencia continua de *sectores* de la pista de un plato. El tamaño de un bloque varía 512 bytes a varios kilobytes. Los datos se transfieren entre el disco y la memoria principal en unidades de bloques.
  - **Bloques más chicos** ⇒ mayor número de transferencias.
  - **Bloques más grandes** ⇒ transferir más de lo realmente requerido.
- El acceso al disco es en varias órdenes de magnitud más lento que el acceso a la memoria principal. Se ha prestado mucha atención a mejorar la velocidad de acceso a bloques de disco.

Optimización en el acceso a los datos en disco

Algunas estrategias:

- *Algoritmos de planificación del brazo del disco*: algoritmo del ascensor.
- *Organización física de los archivos*: organizar físicamente los archivos en bloques contiguos, tal como necesiten accederse (*defragmentación*).
- *Escrituras intermedia en Memoria RAM no volátil*, posterga las escrituras a disco.
- Otros

Sobre el tiempo de acceso

RAID

RAIDs - Redundant Array of Independent Disks - Técnica para organizar y administrar varios discos físicos como si fueran una unidad

+ Aumenta la *capacidad*
+ Aumenta la *velocidad de acceso* a la información usando múltiples discos en paralelo.
+ Aumenta la *confiabilidad*, dando la oportunidad de almacenar datos redundantes, de forma que pueda recuperarse información, aún ante la presencia de fallos.

Conceptos básicos

- **Mirroring o espejado:** enfoque sencillo (pero costoso) para asegurar la información es introducir redundancia duplicando los discos.
  - (-) El proceso de escritura se hace en todas las copias.
  - (+) En caso de falla de un disco, siguen sus copias.

- **Striping:** distribución (segmentado) de datos. Puede hacerse a nivel de bit, (un bit por disco), o nivel de bloque.
  - (+) Mejora el rendimiento mediante paralelismo.
  - (+) Combinado con paridad, mejora la confiabilidad.

RAID

Redundant Array of Independent Disks

- **RAID Level 0**: Block striping; non-redundant.
- **RAID Level 1**: Mirrored disks with block striping.
- **RAID Level 2**: Memory-Style Error-Correcting-Codes (ECC) with bit striping.
- **RAID Level 3**: Bit-Interleaved Parity.
- **RAID Level 4:** Block-Interleaved Parity.
- **RAID Level 5:** Block-Interleaved Distributed Parity.
- **RAID Level 10**: P+Q Redundancy.

Factores para seleccionar nivel RAID

- *Costo* (económico).
- Performance, número de operaciones de I/O por segundo bajo condiciones normales.
- *Performance* del sistema mientras dura la *falla*.
- *Tiempo* requerido para *recuperar* un disco.

- En general se opta entre las opciones RAID 1 y 5

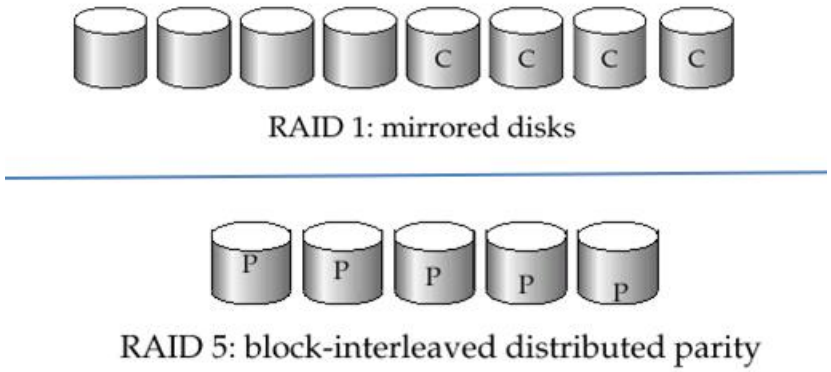
RAID 1: mirrored disks

RAID 5: block-interleaved distributed parity

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

RAID 1 vs. RAID 5

- **R1** presenta mejor performance que **R5**.
  - R5 requiere de 2 búsquedas de bloque (lecturas) y 2 escrituras de bloque para escribir un bloque.
  - R1 requiere 2 escrituras de bloque para escribir un bloque.
- **R1** tiene costo mayor de almacenamiento que **R5**.
- Se prefiere **R5** para aplicaciones con baja tasa de actualización y grandes volúmenes de datos.
- Se prefiere **R1** en cualquier otro caso.



RAID 1: mirrored disks

RAID 5: block-interleaved distributed parity

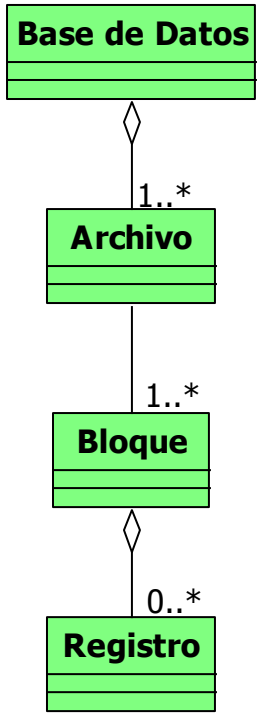
Organización de Archivos y Registros

Distribución de Registros en Almacenamiento

- Estructura general
- Transferencia de Almacenamiento Secundario a Almacenamiento Primario
- Organización de registros en archivos
  - Secuencial
  - Multiclustering
  - Otros

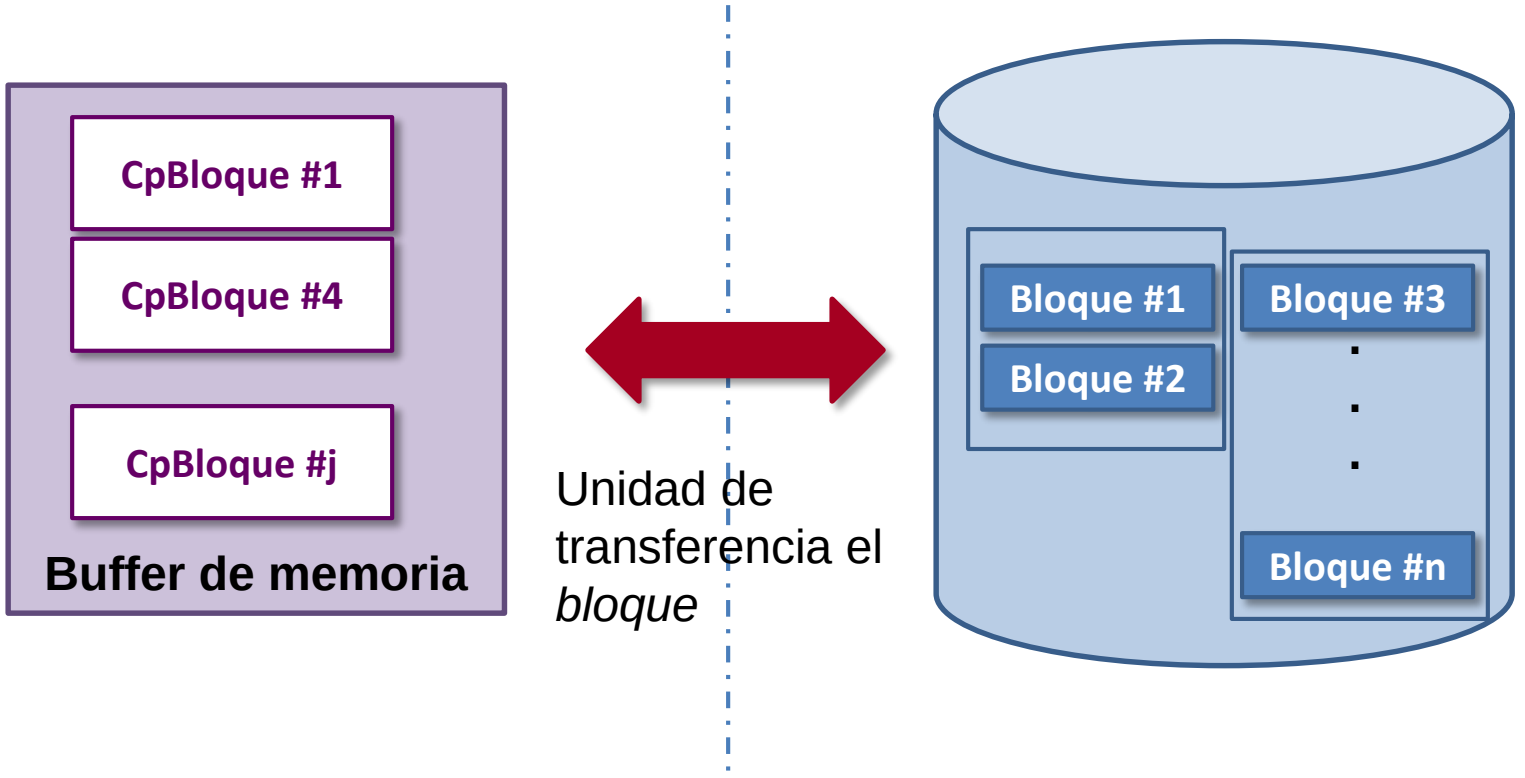
Organización de Archivos

- La base de datos se guarda físicamente en **archivos**. Un archivo se constituye por una secuencia de **registros**.
- Cada archivo se divide en unidades de almacenamiento y transferencia de longitud constante denominadas **bloques**.
- La **memoria intermedia** (buffer) es la parte de la memoria principal disponible para el almacenamiento de las copias de los bloques del disco que se están usando (para leer o escribir).



- **Buffer manager** es el subsistema responsable de asignar espacio del buffer a los bloques requeridos.
- **TIP!** El SMBD busca minimizar las transferencias de bloques entre el disco y la memoria

Transferencia Memoria Disco



Buffer Manager (BM)

- Por *cada solicitud de un bloque de dato* $\mathbf{b_i}$
  1. Si $\mathbf{b_i}$ está en el buffer de memoria, el BM retorna la dirección de $\mathbf{b_i}$ en memoria principal.
  2. Si $\mathbf{b_i}$ no está en el buffer de memoria, el BM debe:
     i. Asignar espacio del buffer para el nuevo bloque:

        Remplazando algún otro bloque, si es necesario hacer espacio, y pero escribiendo previamente el bloque remplazado en disco si es que fue modificado desde la última vez que se copio en disco.

     ii. Leer el bloque desde el disco y transferirlo al buffer de memoria y retornar la dirección de memoria principal que aloja el bloque.

Organización de Archivos

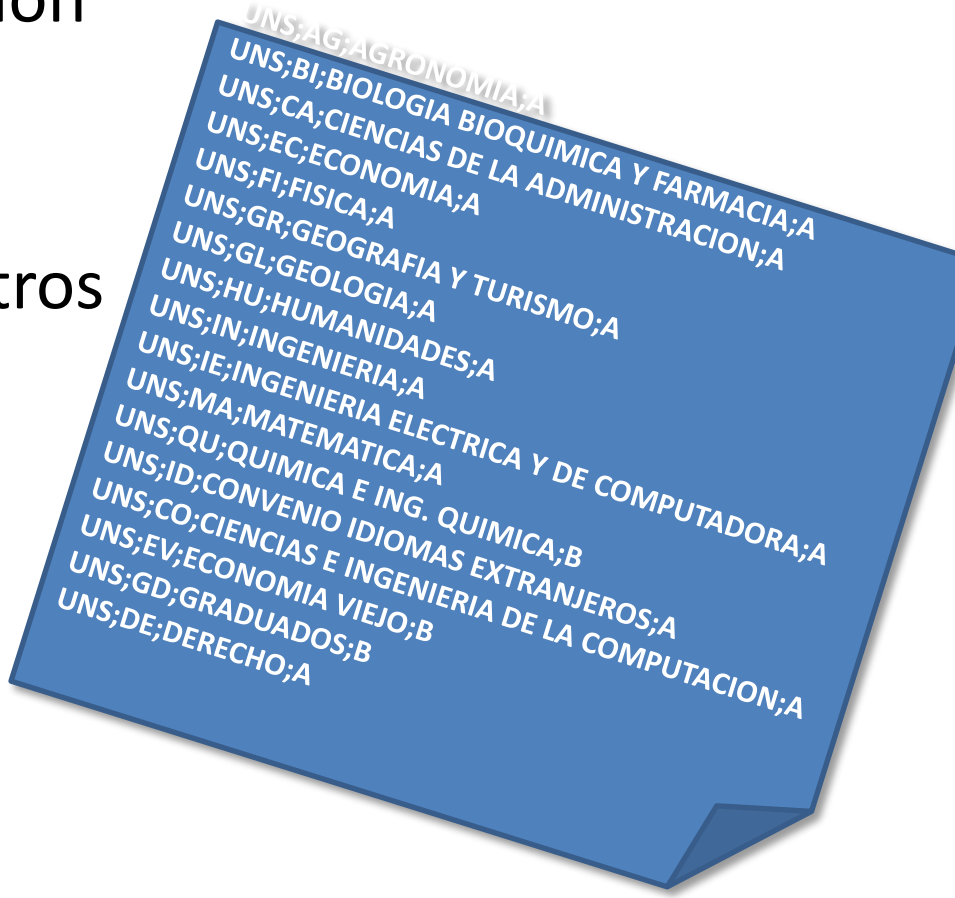
Se denomina **file organization** a la estrategia usada para distribuir registros de un archivo en el dispositivo de almacenamiento secundario.

Algunos DBMS permiten seleccionar y/o personalizar la organización de cada archivo físico según se busque:

- Recuperación de registros rápida.
- Uso eficiente del espacio físico
- Protección de fallas y minimizar pérdidas de datos
- Minimizar los requerimientos de reorganización.

Una organización simple

- Un archivo por cada relación de la base de datos.
- Si se usa el modelo relacional, todos los registros de una relación tienen el mismo tamaño.



Organización de Registros en Archivos

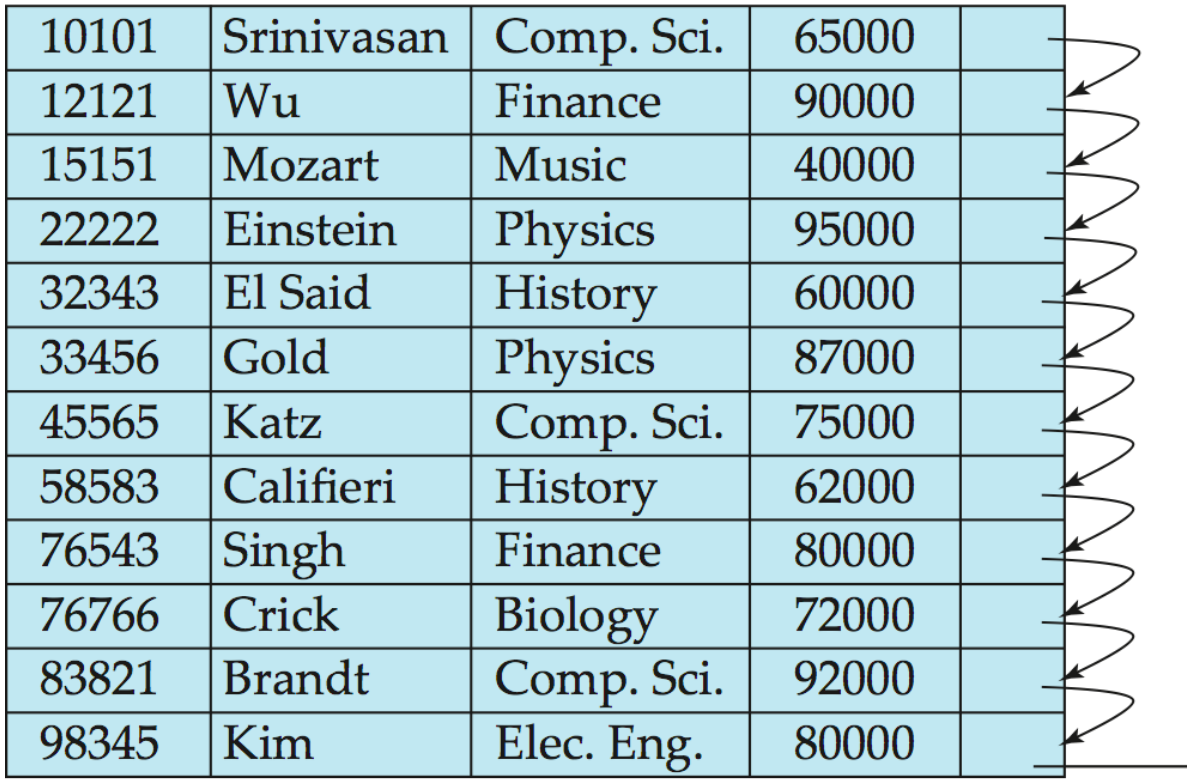
- **Heap:** cada registro en cualquier posición archivo en que haya espacio suficiente. No hay orden entre registros.
- **Secuencial:** los registros se guardan en orden secuencial, por el valor de una *clave*.
- **Hash:** calcula una función de asociación (*hash*) sobre algún/os atributo/s. El resultado *hash* especifica el bloque dónde ubica el registro.
- **Multitable clustering file:** guarda en un archivo registros de varias relaciones diferentes. Favorece el almacenamiento de registros relacionados en un bloque para minimizar I/O.

Organización de Registros Secuencial

- Procesamiento eficiente de los registros de acuerdo con un orden basado en alguna *clave* de búsqueda (no se requiere que sea la clave primaria).
- Los registros se conservan ordenados por medio de punteros. El puntero de *cada registro apunta al siguiente registro según el orden indicado por la clave de búsqueda*.
- Se espera guardar los registros físicamente de acuerdo con el orden indicado o en un orden tan cercano como sea posible.
- Requiere reorganizaciones periódicas.

Organización secuencial

- Los registros se ordenan por algún *atributo clave*.
- Es adecuado para aplicaciones que requieren procesamiento secuencial de archivos.
- Periódicamente requieren de reorganización.



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	

Fuente: Database System Concepts Silberschatz, Korth, Sudarshan

Multitable clustering file

departamentos

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructores

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

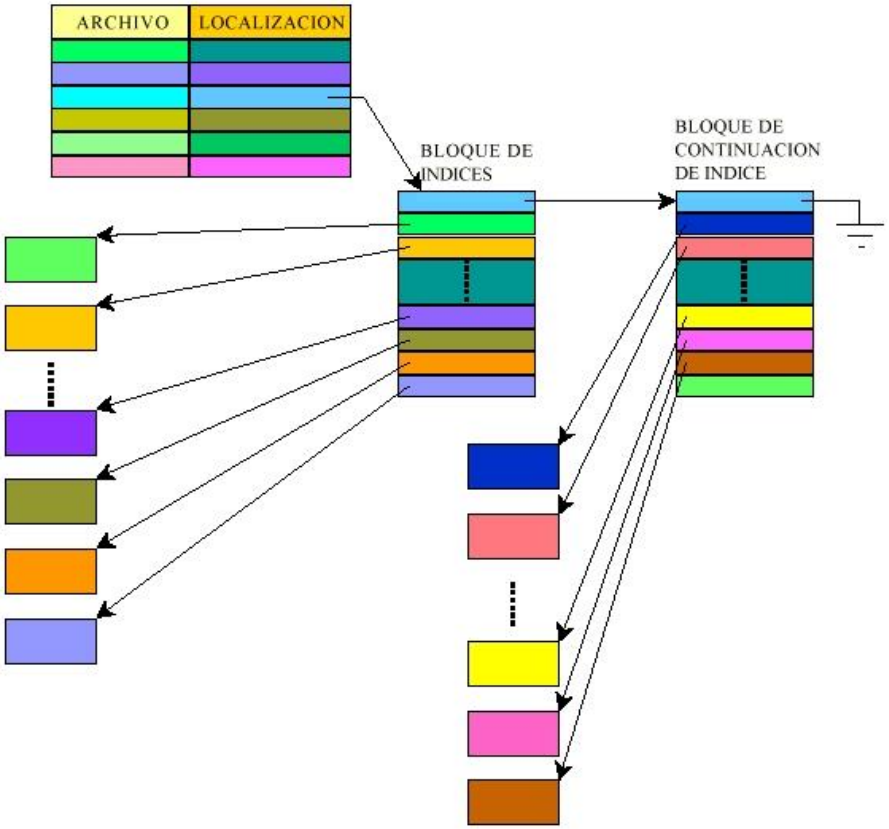
Archivo clustering *departamentos* + *instructores*

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

Fuente: Database System Concepts Silberschatz, Korth, Sudarshan

Hash Estático

- Un **bucket** es una unidad de almacenamiento (típicamente un bloque) que contiene registros.
- Para un **archivo con organización hash,** el bucket que contiene un registro se obtiene calculando una **función hash** sobre las claves de búsqueda.
- Una función hash *h* se define para K valores de búsqueda y el conjunto B de buckets. La misma función h se usa para buscar, insertar o borrar registros.
- Distintos valores de K pueden mapear al mismo bucket. La búsqueda de un registro en un bucket es secuencial.



Figura˜4.5: Encadenamiento de bloques de índices.

Índices

Indexado de Archivos

Indice: “clave” + puntero

- Métricas
- Tipos
  - Ordenados
    - Indice primario – clustering
    - Indice secundario
    - Indice denso e índice ralo.
    - Insertar y borrar usando indices
    - Indices Multinivel – B+Arbol
  - Hash

Índices

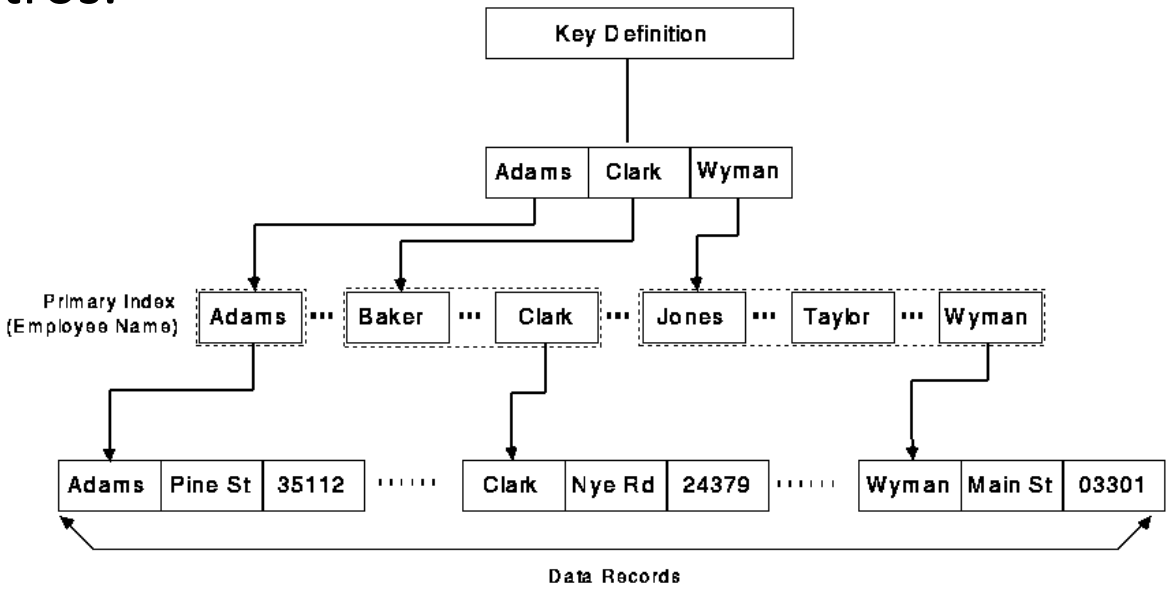
- Los índices son una estructura usada para acelerar el proceso de búsqueda de los registros que cumplan con una *condición de filtro o búsqueda*.
  - Ejemplo: "*cuentas* cuyo número de cuenta está entre 10111 y 10453", el SMBD busca el índice para encontrar el/los bloques de disco donde se encuentran registros de *cuenta* que cumplen la condición

Clave de búsqueda	Puntero

- Archivo índice:
  - **Clave de búsqueda**: uno o más atributos del archivo para búsqueda.
- La lista ordenada de números de cuenta no es "corta" si la base de datos es grande: el propio índice es muy grande y encontrar una cuenta en el índice puede consumir mucho tiempo.

Organización Registros Indexada

- Los **registros de dato** se guardan secuencialmente o no, se crea un índice que permite localizar los registros.
- El índice se usa para determinar la ubicación de los registros que satisfacen alguna condición.
- Cada entrada en un índice "mapea" un valor de clave con uno o más registros.



Tipos de índices

- **Índices ordenados.** Basados en una disposición ordenada de los valores.

- **Índices hash**. Basados en la distribución uniforme de los valores en *buckets*.

Métricas para evaluar índices

- **Tipo de acceso.** Según soporten eficazmente búsqueda por igualdad y/o búsquedas por rango.
- **Tiempo de acceso.** El tiempo para encontrar el/los datos requeridos.
- **Tiempo de inserción.** El tiempo requerido para insertar un nuevo dato. Incluye el tiempo necesario para buscar dónde insertarlo y el tiempo para actualizar la estructura del índice.
- **Tiempo de borrado.** El tiempo requerido para borrar un elemento de datos. Incluye el tiempo necesario para buscarlo y el tiempo para actualizar la estructura del índice.
- **Espacio adicional requerido.** El espacio adicional ocupado por la estructura del índice.

Índices Ordenados

- El índice se asocia con una *clave_de_búsqueda*. El **archivo índice** almacena de manera ordenada los valores de las claves de búsqueda, y enlaza con cada clave de búsqueda los registros de datos que contienen esa clave de búsqueda.
- Si el archivo de datos está ordenado secuencialmente, el índice cuya clave de búsqueda especifica el orden secuencial del archivo es el **índice primario.**
  - El índice primario **usualmente** es la clave primaria.
  - Los índices que tienen el mismo orden secuencial que los registros de datos se denominan **índices clustering**
- Los índices cuyas claves de búsqueda especifican un orden diferente del orden secuencial del archivo se llaman **índices secundarios** o **no clustering.**

Índices Densos – Índices Ralos

Registro índice = clave_de_búsqueda + puntero/s

Puntero = identificador_de _bloque_de _disco + desplazamiento.

Dos clases de índices ordenados:

- **Índice denso:** un registro índice **por cada valor** de *clave_de_búsqueda* en el archivo.

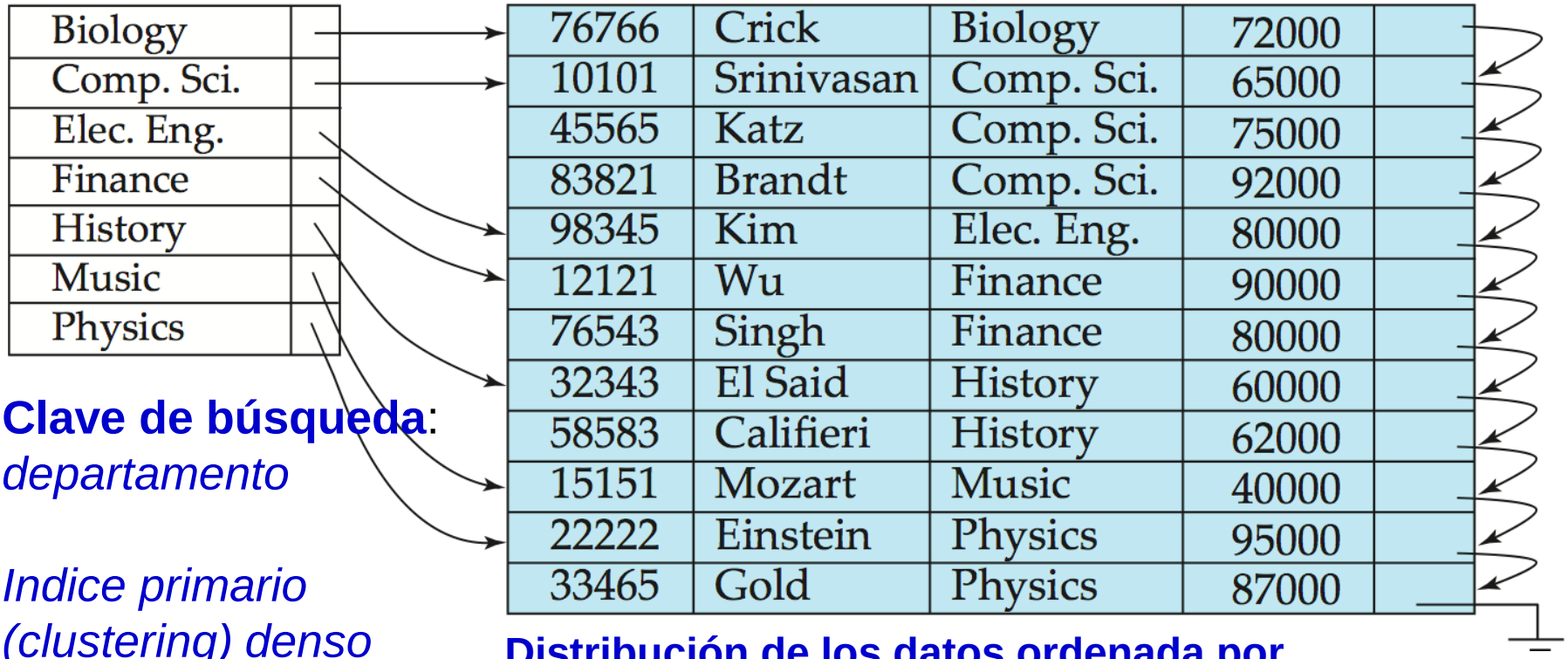
Registro índice = clave_de_búsqueda + puntero al primer registro con valor de la clave.

- **Índice ralo (sparse):** el índice tiene los registros **para ciertos valores** de la *clave_de_búsqueda*.

Registro índice = clave_de_búsqueda + puntero al primer registro con valor igual al valor de la clave.

Para localizar un registro se busca la entrada del índice con el valor más grande que sea menor o igual que el valor que se está buscando. Se sigue el puntero al archivo hasta encontrar el registro deseado.

Índice Denso



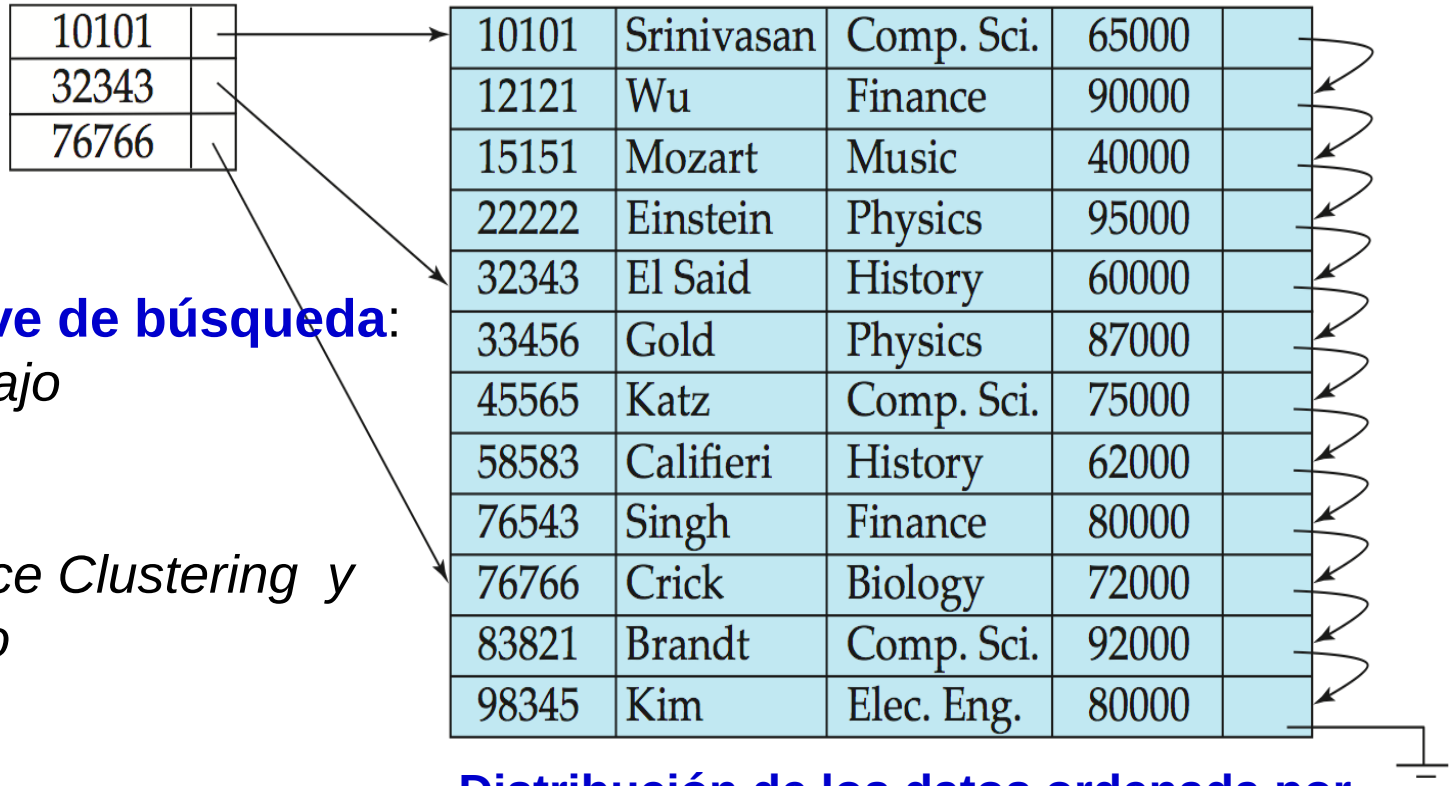
Clave de búsqueda:
departamento

Indice primario (clustering) denso

Distribución de los datos ordenada por departamento

Fuente: Database System Concepts Silberschatz, Korth, Sudarshan

Índice Ralo



Clave de búsqueda: *Legajo*

Indice Clustering y Ralo

Distribución de los datos ordenada por legajo

Fuente: Database System Concepts Silberschatz, Korth, Sudarshan

Índices Densos vs. Índices Ralos

- El índice ralo requiere menos espacio y menos overhead por tareas ingresar/borrar registros.
- El índice ralo es más lento que el índices denso para encontrar un registro.
- El índice ralo solo se puede usar cuando el archivo de datos ordenados como la clave del índices (indice primario o clustering).
- **La solución de compromiso**: índice ralo con una entrada por bloque del archivo de datos, que se corresponda con la clave de búsqueda para el bloque

Actualizar el índice – Insertar en el índice Denso

- Si *clave_de_búsqueda* no aparece en el índice, insertar un registro índice con el valor de clave_de_búsqueda en la posición adecuada y apuntar al registro de dato.
- Si *clave_de_búsqueda* aparece en el índice:
  - Si el registro índice almacena punteros a todos los registros con *clave_de_búsqueda*, añadir el puntero al nuevo registro en el registro índice.
  - Sino, (se almacena un puntero sólo hacia el primer registro de clave_de_búsqueda) situar el registro insertado después de los otros con los mismos valores.

Insertar índice Ralo

Se asume que el índice almacena una entrada por cada bloque del archivo de datos.

- Si por la inserción se crea un bloque nuevo, insertar en orden en el archivo índice el primer valor de *clave_de_búsqueda*.
- Si el nuevo registro tiene el menor valor de *clave_de_búsqueda* en su bloque, actualizar la entrada del índice que apunta al bloque.
- Sino, no se requieren cambios en el índice.

Borrar – índice denso

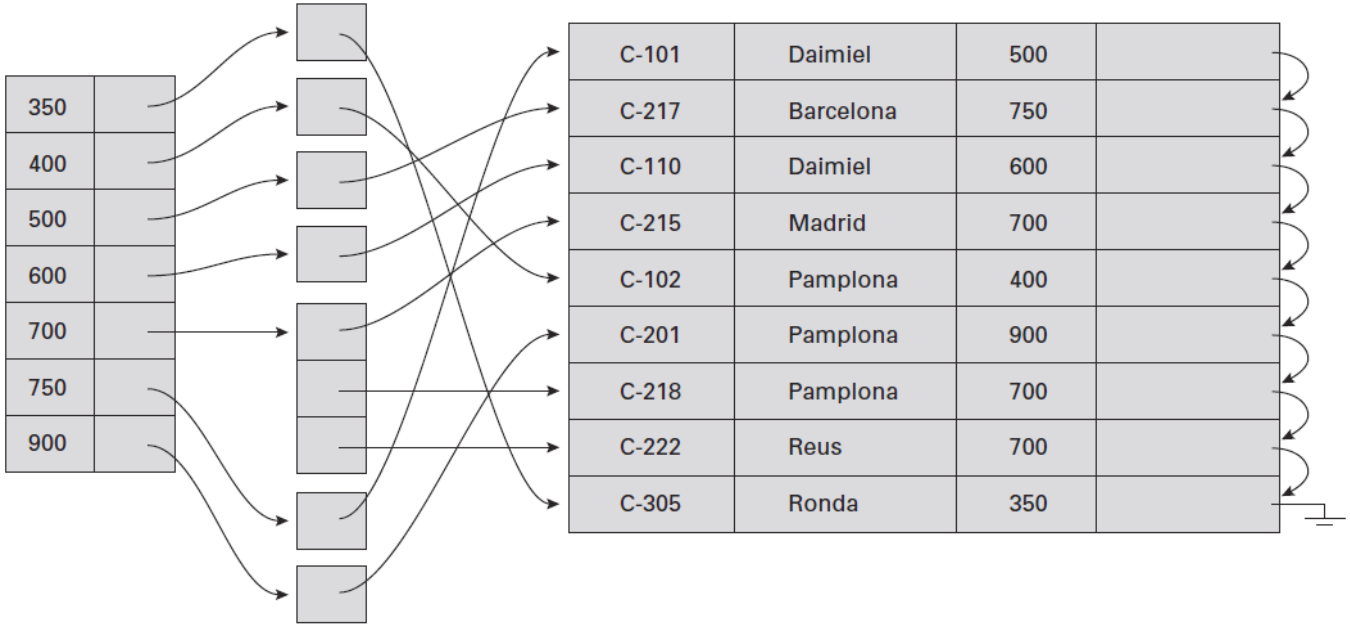
1. Buscar el registro índice a borrar.
2. Si el registro borrado era el único registro con *clave_de_búsqueda*, borrar el registro índice.
3. Sino:
   i. Si el registro índice almacena punteros a todos los registros de clave_de_búsqueda, borrar el puntero al registro borrado.
   ii. Sino, si el registro borrado era el primer registro con el valor de la clave de búsqueda, actualizar el registro índice para apuntar al siguiente registro.

Borrar – Índice Ralo

- Si el índice no contiene un registro índice con *clave_de_búsqueda*, finalizar.
- Sino
  - Si el registro borrado era el único registro con la clave de búsqueda, cambiar el registro índice correspondiente con el siguiente valor de *clave_de_búsqueda*. Si el siguiente valor de la clave de búsqueda ya tiene una entrada en el índice, se borrar el registro índice.
  - Sino, actualizar el registro índice para que apunte al siguiente registro *clave_de_búsqueda*.

Índices Secundarios

- **Los índices secundarios** sirven para recorrer los datos en un orden distinto al orden físico.
- Los índices secundarios siempre deben ser densos y con una entrada en el índice por cada valor de la clave de búsqueda, y un puntero a cada registro del archivo.
  - No sería suficiente apuntar sólo al primer registro de cada valor de la clave. El resto de registros con el mismo valor de la clave de búsqueda podrían estar en cualquier otro sitio del archivo



Índices Secun-darios

- Se puede usar un **nivel adicional de indirección** para implementar índices secundarios sobre claves de búsqueda que no son claves candidatas.
- Los punteros en estos índices secundarios no apuntan directamente al archivo. En vez de eso, *cada puntero apunta a un cajón que contiene punteros al archivo*.

Índices Primarios y Secundarios

- Los índices ofrecen beneficios importantes cuando se busca a determinados registros.
- Sin embargo, mantener actualizados los índices imponte un overhead adicional, cada vez que la base de datos es alterada, todos los índices deben ser controlados y si corresponde actualizados.
- La recorrida secuencial de una base siguiendo el índice primario es eficiente, pero siguiendo el índice secundario no es así.

Ejemplo

Sean las relaciones

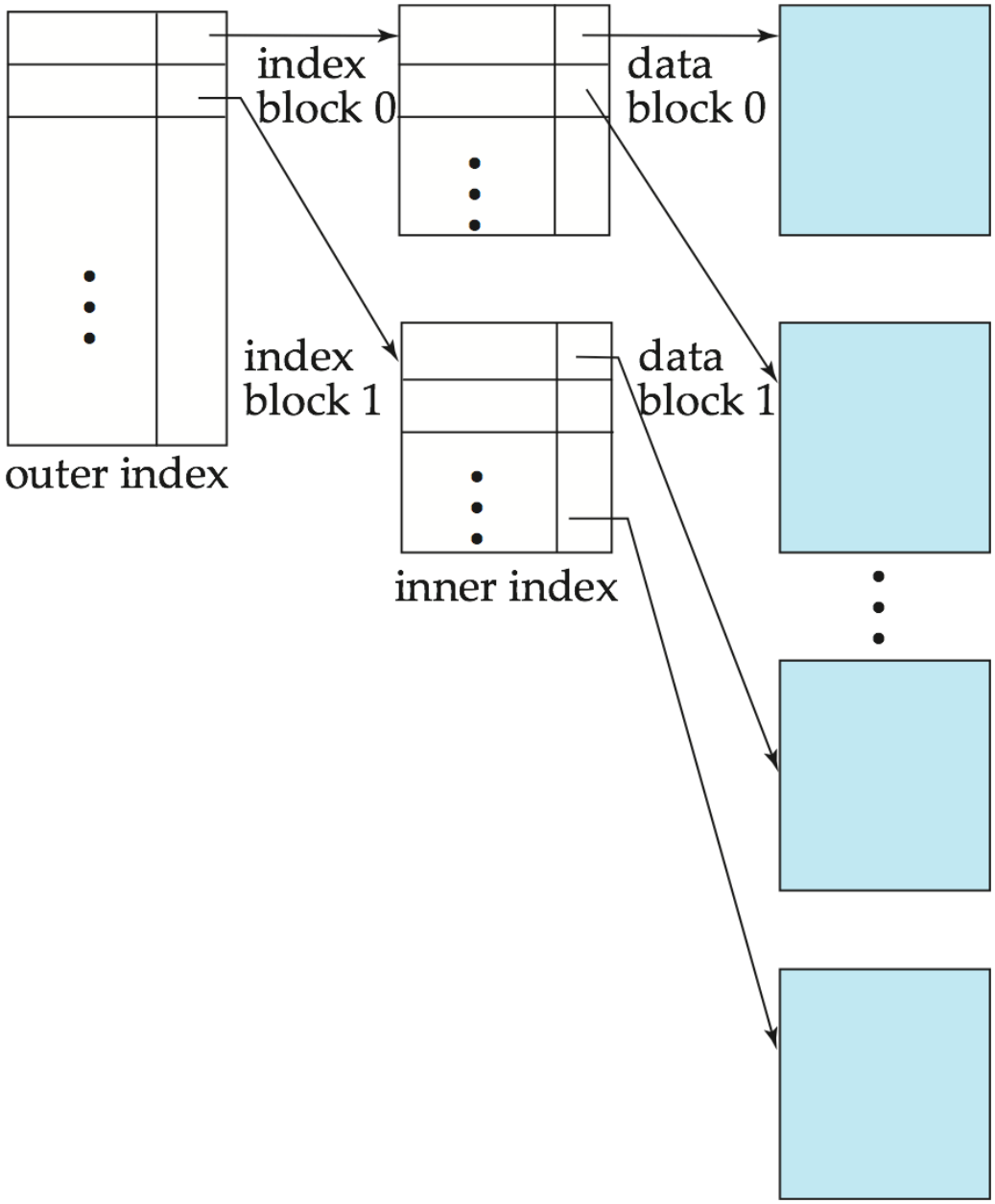
- *instructor* (legajo, nombre, id_dpto, salario)
  - Indice primario: pk_inscructor (legajo)
  - Indices secundario: sec_dpto (id_dpto),
    sec_nombre (nombre)
- *departamentos* (id_dpto, descripcion, edificio)
  - Indice primario: pk_dptos (id_dpto)

- La relación instructores tiene un alto porcentaje de actualizaciones.
- Frecuentemente se consulta los dado un instructor, el edificio donde está ubicado

Índices Multinivel

- Para bases de datos grandes el mismo archivo índice podría ser demasiado grande para mantenerse en memoria y permitir un procesamiento eficiente.
- Así los índices de mayor tamaño se almacenan como archivos secuenciales en disco. Buscar una entrada implicará leer varios bloques de disco.
- **Solución**: tratar al índice como si fuese un archivo secuencial y se construir un índice ralo sobre el índice primario.

Índices Multinivel

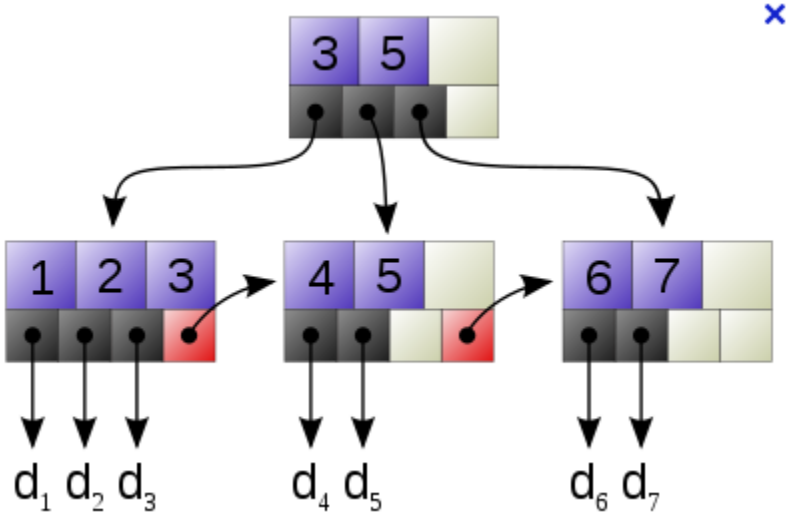


Fuente: Database System Concepts
Silberschatz, Korth, Sudarshan

Localizar un registro

1. Realizar una búsqueda binaria en el índice externo buscando el registro con el mayor valor de *clave_de _búsqueda* menor o igual al valor deseado.
2. Seguir el puntero que apunta al bloque índice de nivel dos.
3. Recorrer el bloque hasta encontrar el registro con el mayor valor de clave que sea menor o igual que el valor deseado. El puntero de este registro apunta al bloque del archivo buscado.

- Con dos niveles de indexación y el índice externo en memoria principal, se necesita leer un único bloque índice.
- Los índices con dos o más niveles se llaman índices **multinivel**.



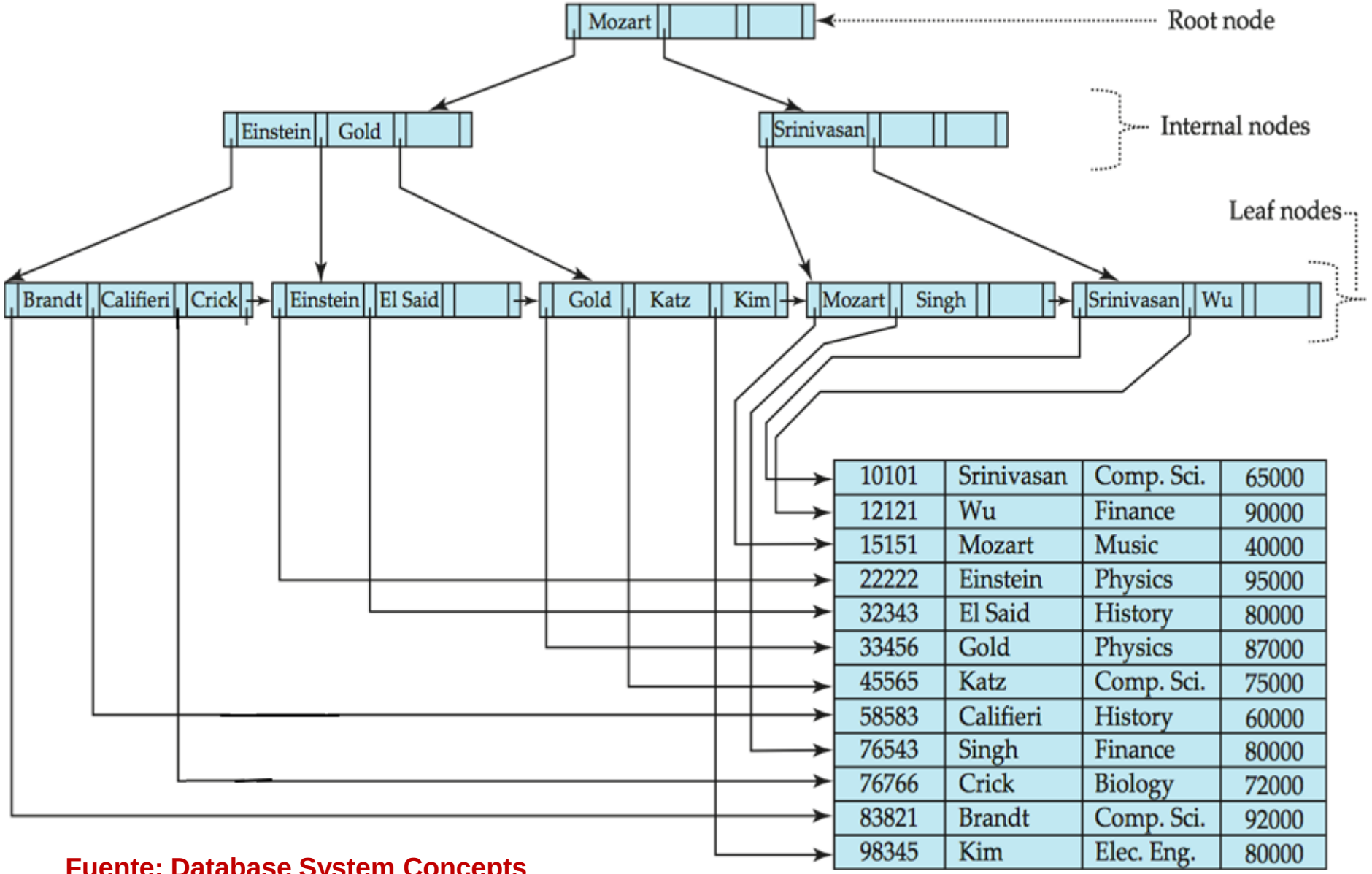
B+ Árboles

Un índice de árbol B+ es un índice multinivel con una estructura que difiere del índice multinivel de un archivo secuencial

Índices B+ Árbol

- Un índice B+ árbol es un **árbol balanceado** donde los caminos desde la raíz a cada hoja del árbol son de la misma longitud.
- Cada nodo que no es una hoja tiene entre $\lceil n/2 \rceil$ y n hijos, con n es fijo para cada árbol. El nodo raíz es un caso especial
- Un nodo típico de un árbol B+ contiene hasta $n - 1$ claves de búsqueda $K_1, K_2, ..., K_{n-1}$ y n punteros $P_1, P_2, ..., P_n$.
  - K_i claves de búsqueda, $K_1 < K_2$ ...
  - P_i punteros a hijos, para nodos intermedios o punteros a registros o buckets para nodos hijos

P_1	K_1	P_2	K_2	...	P_{n-1}	K_{n-1}	P_n



Fuente: Database System Concepts
Silberschatz, Korth, Sudarshan

Nodos del B+

- **Nodos intermedios:**
  - P_1 apunta al subárbol que contiene claves menores a K_1.
  - P_2 apunta al subárbol que contiene claves mayores o iguales a K_1 y menores a K_2.
  - ...
  - P_n apunta al subárbol que contiene claves mayores o iguales a K_{n-1}
- **Nodos hoja:**
  - P_i apunta al registro con clave K_i.
  - P_n apunta a la próxima hoja.

B+ Árboles

Dado un B+ Árbol con **n = 6**

- Los **nodos hoja** tienen entre $\lceil (n-1)/2 \rceil$ and n-1, valores (entre 3 y 5 valores para n = 6).
- Los **nodos intermedios** (no raíz) deben tener $\lceil (n/2 \rceil$ and n hijos (con entre 3 y 6 hijos n =6).
- La **raíz** debe tener por lo menos dos hijos

- Una característica de los índices B+ es que se mantienen balanceados. Esto es, el camino desde la raíz a cada nodo hoja es la misma. Esto asegura un buen rendimiento para las búsquedas, inserciones y borrados.

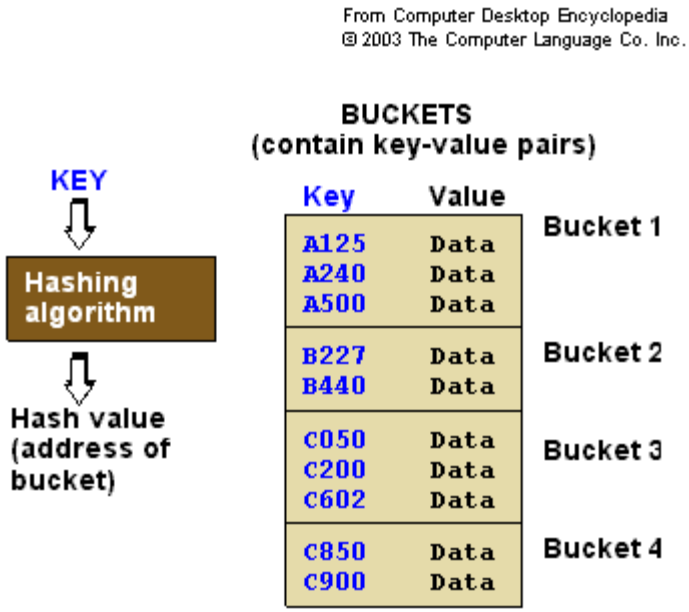
Consultas con B+ Árbol

- Si existen *K* valores para clave de búsqueda en el archivo, el camino no será más largo que $\log_{n/2}(K)$
- Generalmente, cada nodo tiene el mismo tamaño que un bloque de disco.
- Para una **clave de búsqueda de 12 bytes** y un **tamaño del puntero a disco de 8 bytes**, *n* está alrededor de 200. Una estimación más conservadora *n* está en torno a 100.
  - Con *n* = 100, si se tienen un millón de valores de clave de búsqueda en el archivo, una búsqueda necesita solamente 4 accesos a nodos.
- Normalmente, el nodo raíz del árbol se guarda en memoria intermedia; así se necesitan tres o menos lecturas del disco.

Actualizaciones con B+ Árbol

- El borrado y la inserción son más complejos, ya que podría ser necesario **dividir** un nodo que resultara demasiado grande como resultado de una inserción, o **fusionar** nodos si un nodo se volviera demasiado pequeño.
- También se debe asegurar que el equilibrio del árbol se mantiene, con lo cual la operación se puede expandir a otros niveles.

Organización Hash



La Función Hash

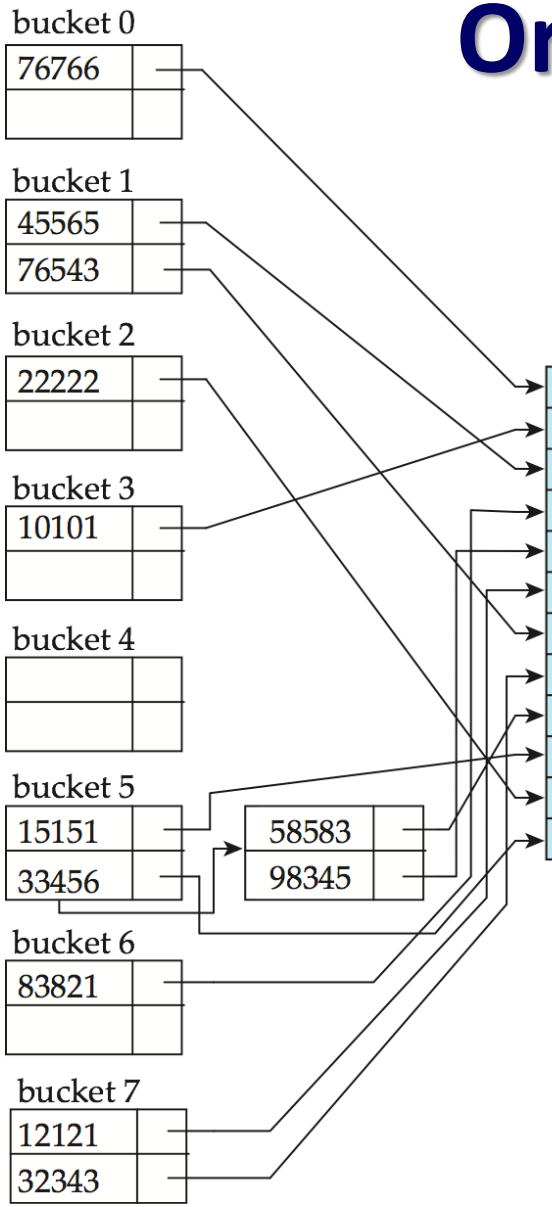
Una **función hash** debería cumplir con las siguientes propiedades:

- Distribución *uniforme*. Esto es, que cada bucket tenga asignado el mismo número de valores de la clave de búsqueda dentro del conjunto de *todos* los valores posibles de la clave de búsqueda.
- Distribución *aleatoria*. Esto es, cada bucket tendrá casi el mismo número de valores asignados a él, sin tener en cuenta la distribución actual de los valores de la clave de búsqueda (por ejemplo, alfabético).

Índices Hash

- Un **índice hash o índice asociativo** organiza las claves de búsqueda, con sus punteros asociados, dentro de una estructura de archivo hash.
  - Se aplica una función de asociación sobre la clave de búsqueda para identificar un cajón,
  - Se almacenan la clave y los punteros asociados en el bucket.
- Estrictamente, los índices hash son siempre índices secundarios.

Organización con índice Hash



76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
33465	Gold	Physics	87000

hash index on *instructor,* on attribute *ID*

Fuente: Database System Concepts
Silberschatz, Korth, Sudarshan

Problemas Hash Estático

- Una función hash basada en el tamaño actual del archivo se degrada a medida que la base de datos crece.
- Una función hash basada en el tamaño futuro, inicialmente pierde una cantidad significativa de espacio.
- Reorganizar implica elegir una nueva función hash, volver a calcular la función de cada registro y reasignar ubicaciones. Esto es una operación masiva que requiere mucho tiempo. Además, es necesario prohibir el acceso al archivo durante la reorganización.
- Existen otras propuestas que trabajan con **hash dinámico**.

Resumiendo

- Existen distintas formas para organizar los archivos e índices. Cada una tiene sus ventajas y desventajas.
- Algunos DBMS (no todos) proveen alternativas, dejando al diseñador de la base de datos la decisión final.
- Se deben considerar los siguientes aspectos:
  - Costo de una reorganización periódica del índice.
  - Frecuencia de las actualizaciones.
  - Tiempo promedio de acceso.
  - Tipos de consultas se supone que van a realizar los usuarios

Ejemplo

- En relación a las consultas se puede decir que:
  - Si predominan las consultas por igualdad (por ejemplo "**SELECT * FROM r WHERE r.c='c1'** "), bajo una adecuada función hash, los esquemas hash tienen mejor comportamiento.
  - Si predominan las consultas por rango (por ejemplo "**SELECT * FROM r WHERE r.c >'c1'** ") las propuestas ordenadas tienen mejor tiempo de respuesta.

Cuándo usar índices

Trade-off entre “mejora la performance en las consultas” y el “costo de las actualizaciones”. Algunas reglas “intuitivas”

- Los índices son útiles en tablas de muchos registros.
- Crear el índice para la clave primaria de una tabla.
- Crear índices secundarios sobre las columnas que aparecen con frecuencia en la sección WHERE y SELECT de consultas SELECT.
- Idem para columnas que aparecen en la sección ORDER BY y GROUP BY (asegurarse que el DBMS usa índices para este tipo de consultas).
- Crear índices cuando los valores distintos del atributos son significativos.
- Si los atributos que participan del índice admiten valores nulos, investigar el comportamiento del índice con atributos nulos.

Temas de la clase de hoy

- Conceptos de Normalización de almacenamientos
  - Dependencias funcionales.
  - Claves de almacenamiento.
  - Atributos primos y no primos.
- Formas normales (1FN, 2FN, 3FN y FNBC)

- **Bibliografía**:
  - *Database System Concepts* – Abraham Silberschatz – Capítulo 10 y 11
  - *DataBase System – The Complete Book* – H. Molina, J. Ullman. Capítulo 14.