

CONCEPTOS DE BASES DE DATOS

CLASE 4



- **Búsqueda de información en aplicaciones antiguas**
 - Se intenta lograr la **menor competencia posible** por el acceso a los archivos maestro, almacenados en cintas con acceso secuencial
 - Las aplicaciones mantienen cierta redundancia de datos en archivos propios y es imperioso un **acceso eficiente** a los datos almacenados
 - Búsqueda de datos con la menor cantidad de desplazamientos en disco → **estudio de performance**

- **Estudio de performance**

- La **búsqueda secuencial** se puede tomar como una referencia de comparación al momento de realizar estudios de **performance de acceso a los datos**
- Costo de acceso
 - N° de comparaciones → operaciones en memoria primaria
 - N° accesos a disco → operaciones en memoria secundaria
- Casos de análisis
 - Mejor caso
 - Peor caso
 - Caso promedio

- **Estudio de performance**

- **Búsqueda secuencial** (en un archivo con **N** registros)
 - Mejor caso: leer **1** registro (0 si esta en buffer)
 - Peor caso: leer **N** registros
 - Caso promedio: leer **N/2** registros
 - Orden: **O(N)** → depende de la cantidad de registros
- Lectura de bloques de registros
 - Ahorra tiempo → disminuye el número de desplazamientos, pero sigue siendo de **O(N)**

- Búsqueda de un registro por **NRR**
 - Rápido → Acceso directo → **$O(1)$**
 - Solo sirve si es posible obtenerlo y se conoce
- Búsqueda de un registro **secuencialmente**
 - Archivo desordenado
 - Es posible recorrer todo el archivo sin éxito: **$O(N)$**
 - Archivo ordenado
 - Se detiene al encontrar un registro “mayor”: **$O(N)$**
- Alternativa → búsqueda **binaria**

- Búsqueda **binaria**
 - Para poder aplicarla:
 - Archivo con registros de longitud fija
 - Archivo ordenado por clave
 - Pasos:
 - Se compara la clave buscada con la encontrada en el registro que se ubica en la **mitad del archivo**
 - Si no se encuentra el elemento buscado, se continúa procesando la **mitad del archivo que corresponda**

Archivos

Búsqueda de información

- Búsqueda **binaria**
 - Se busca el elemento **87**:

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1 | 3 | 4 | 5 | 8 | 11 | 15 | 18 | 21 | 25 | 29 | 30 | 40 | 43 | 45 | 56 | 67 | 87 | 88 | 90 |
|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1 | 3 | 4 | 5 | 8 | 11 | 15 | 18 | 21 | 25 | 29 | 30 | 40 | 43 | 45 | 56 | 67 | 87 | 88 | 90 |
|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1 | 3 | 4 | 5 | 8 | 11 | 15 | 18 | 21 | 25 | 29 | 30 | 40 | 43 | 45 | 56 | 67 | 87 | 88 | 90 |
|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

Archivos

Búsqueda de información

*{Nota: la función devuelve el **NRR** del registro encontrado o -1 si no está en el archivo}*

FUNCTION busqBinaria(archivo, clave)

BEGIN

registro = -1; menor = 1; mayor = **SIZE**(archivo);

MIENTRAS (menor <= mayor) && (registro == -1)

{Se evalúa el registro que se encuentra en el medio del archivo}

medio = (mayor+menor)/2

POSICIONARSE(archivo, medio)

LEER_REG(archivo, reg)

claveReg = **CONSTRUIR_FC**(reg)

{Si el valor encontrado es mayor al buscado → sigo con la mitad inferior}

SI (claveReg > clave) **ENTONCES** mayor = medio – 1

{Si el valor encontrado es menor al buscado → sigo con la mitad superior}

SINO SI (claveReg < clave) **ENTONCES** menor = medio + 1

{En c.c., el registro encontrado es el que se buscaba}

SINO registro = medio

FIN MIENTRAS

DEVOLVER(registro)

END

- Búsqueda **binaria**
 - Siendo N = # de registros $\rightarrow O(\log_2 N)$
 - **Ventajas**
 - En cada paso acota el espacio para encontrar la información
 - Se mejora la performance de la búsqueda secuencial
 - **Desventajas**
 - Costo \rightarrow mantener el archivo ordenado

- Es necesario mantener el archivo ordenado para poder utilizar la búsqueda binaria
- **Algoritmos de clasificación (ordenación)**
 - Existen diferentes enfoques para clasificar un archivo
 - Se analizarán algunas de las técnicas existentes (en la bibliografía se pueden encontrar los algoritmos correspondientes)
 - Clasificación en disco
 - Clasificación del archivo completo en RAM
 - Clasificación de las claves en RAM
 - Clasificación por partes

- **Clasificación en disco**

- Todo algoritmo de clasificación **requiere recorrer el archivo en más de una oportunidad**, realizando comparaciones y reorganizando sus elementos → $O(N^2)$
- Ordenar un archivo en disco involucra saltos, desplazamientos y relectura de datos, por lo que resulta **demasiado lento** para archivos de un tamaño importante.

- **Clasificación en disco**

- Ej: algoritmo **ordenamiento de burbuja**. Consiste en comparar elementos adyacentes e intercambiarlos en el caso que el orden no es el correcto.

| Archivo | 5 | 3 | 2 | 6 | 4 | 1 |
|---------|---|---|---|---|---|---|
| Paso 1 | 3 | 2 | 5 | 4 | 1 | 6 |
| Paso 2 | 2 | 3 | 4 | 1 | 5 | 6 |
| Paso 3 | 2 | 3 | 1 | 4 | 5 | 6 |
| Paso 4 | 2 | 1 | 3 | 4 | 5 | 6 |
| Paso 5 | 1 | 2 | 3 | 4 | 5 | 6 |

- **Clasificación del archivo completo en RAM**
 - Consiste en llevar el archivo a memoria RAM y luego ordenarlo
 - Se llevan todos los registros del archivo a un arreglo en memoria RAM
 - Se genera un segundo arreglo con las claves en forma canónica y luego se ordena por clave
 - Estructura: (claveFC, NRR)
 - Basándose en el orden logrado, se reescribe el archivo en almacenamiento secundario de forma ordenada

- **Clasificación del archivo completo en RAM**
 - El archivo a ordenar **debe caber completamente** en la memoria RAM
 - Sólo sirve para archivos pequeños
 - Se debe encontrar una alternativa que permita ordenar archivos grandes
 - No reordenar el archivo completo en memoria

- **Clasificación de las claves en RAM**
 - No se lleva el archivo completo a memoria, sino que **sólo se llevan las claves**
 - Permite ordenar archivos más grandes
 - Una vez realizada la clasificación, existe la necesidad de **releer** el archivo completo y **escribirlo** de forma ordenada
 - Nueva cota → **las claves del archivo deben caber completamente** en la memoria RAM

- **Clasificación por partes**
 - Se puede dar el caso de que **ni siquiera las claves de un archivo caben completamente** en la memoria RAM
 - La clasificación debe realizarse **por partes**. Se debe entonces:
 - Partir el archivo
 - Ordenar cada parte
 - Juntar las partes ordenadas

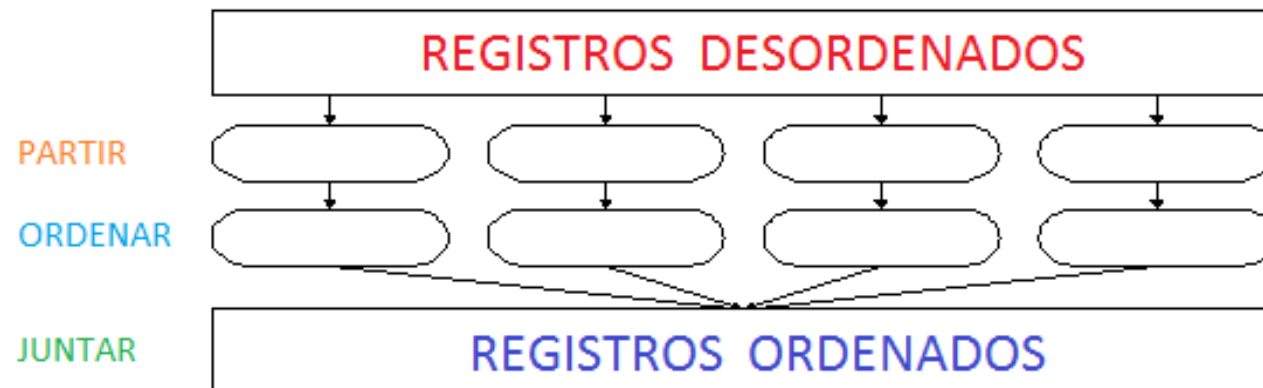
- **Clasificación por partes**
 - Permite clasificar archivos de **cualquier tamaño**
 - Archivo más grande → mayor cantidad de particiones
 - Se debe intentar **minimizar los desplazamientos** realizando procesamientos secuenciales
 - Alternativas para generar las particiones ordenadas
 - Sort interno
 - Selección por reemplazo
 - Selección natural

- **Clasificación por partes: Sort interno**
 - Es la técnica más simple
 - El tamaño de cada partición se determina según el tamaño de la RAM → el máximo posible
 - Ejemplo: archivo de 400.000 registros
 - Tamaño registro: 100 bytes → Tamaño de archivo: 40 MB
 - Tamaño clave: 10 bytes → Tamaño total de claves: 4 MB
 - Tamaño memoria: 1 MB → No entra el archivo ni las claves completamente

- Clasificación por partes: **Sort interno**

- Ejemplo: archivo de 400.000 registros

- Tamaño memoria: 1 MB, Tamaño registro: 100 bytes →
entran en memoria 10.000 registros juntos
- Tamaño partición: 10.000 reg. → 40 particiones



- Clasificación por partes: **Sort interno**
 - El **punto crítico** al intercalar muchos archivos intermedios en disco es el **tiempo de desplazamiento**, que depende de:
 - Cantidad de particiones que se intercalan
 - Espacio en memoria para almacenar partes de las particiones
 - Intercalación en el ejemplo anterior
 - Se debe leer datos de las **40 particiones simultáneamente**, se podrá leer **1/40 de cada partición** → para leer una partición completa se necesita **40 desplazamientos**
 - Como hay 40 particiones → $40 * 40 = 1600$ **desplazamientos** → **EXCESIVO**

- **Clasificación por partes: Sort interno**
 - En general, para la intercalación de **k particiones**, donde cada partición es del tamaño de RAM disponible:
 - Se requieren **k desplazamientos** para leer todos los registros en cada partición.
 - Como hay **k** particiones, la intercalación completa requiere **k² desplazamientos**
 - La clasificación e intercalación es de **O(k²)**. Como **k** es proporcional a **N** → **O(N²)**

- **Clasificación por partes**
 - ¿Cómo mejorar la performance del Sort interno?
 - Se necesita **reducir la cantidad de desplazamientos**
 - Mayor tamaño de partición → menor cantidad de particiones
→ menor cantidad de desplazamientos
 - Pero el tamaño de las particiones está acotado por el espacio disponible en RAM. Opciones:
 - Disponer de una memoria RAM más grande
 - Utilizar una técnica que genere particiones más grandes
 - Utilizar un método de intercalación más eficiente

- **Clasificación por partes: Sel. por reemplazo**
 - Aumenta el tamaño de las particiones al **doble** (promedio)
 - Pasos:
 1. Leer M registros desordenados (tantos como quepan) e iniciar una **nueva partición**
 2. Obtener **clave menor**
 3. Pasar la **clave menor** al **archivo de salida** (alm. secundario)
 4. Reemplazar por otro, si tiene clave menor a la pasada al archivo de salida → **dormirlo**
 5. Repetir desde el paso 2 hasta que todos los registros en la entrada estén dormidos
 6. Comenzar con una **nueva partición** despertando todos los dormidos, desde el paso 2 y hasta terminar el archivo original

- **Clasificación por partes: Sel. por reemplazo**

- Claves en alm. secundario: 17, 48, 6, 13, 68, 22, 8, 18, 15, 59, 25, 19, 34. Se inicia la 1° partición:

| Resto de la entrada | Mem. principal | Partición generada |
|---------------------------------------|----------------|-----------------------|
| 34, 19, 25, 59, 15, 18, 8, 22, 68, 13 | 6 48 17 | - |
| 34, 19, 25, 59, 15, 18, 8, 22, 68 | 13 48 17 | 6 |
| 34, 19, 25, 59, 15, 18, 8, 22 | 68 48 17 | 13, 6 |
| 34, 19, 25, 59, 15, 18, 8 | 68 48 22 | 17, 13, 6 |
| 34, 19, 25, 59, 15, 18 | 68 48 (8) | 22, 17, 13, 6 |
| 34, 19, 25, 59, 15 | 68 (18) (8) | 48 ,22, 17, 13, 6 |
| 34, 19, 25, 59 | (15) (18) (8) | 68, 48, 22, 17, 13, 6 |

- 1° partición completa, ya que todos los registros en memoria están dormidos. Tamaño: 6 registros.

- **Clasificación por partes: Sel. por reemplazo**
 - Claves que quedan en memoria secundaria: 59, 25, 19, 34. Se inicia la 2° partición:

| Resto de la entrada | Mem. principal | Partición generada |
|---------------------|----------------|---------------------------|
| 34, 19, 25, 59 | (15) (18) (8) | – |
| 34, 19, 25 | 15 18 59 | 8 |
| 34, 19 | 25 18 59 | 15, 8 |
| 34 | 25 19 59 | 18, 15, 8 |
| | 25 34 59 | 19, 18, 15, 8 |
| | – 34 59 | 25, 19, 18, 15, 8 |
| | – – 59 | 34, 25, 19, 18, 15, 8 |
| | – – – | 59, 34, 25, 19, 18, 15, 8 |

- 2° partición completa. Tamaño: 7 registros.

- **Clasificación por partes: Selección natural**
 - Es una variante que **reserva y utiliza un buffer en memoria secundaria** → ahorra memoria RAM
 - Los **dormidos** se pasan a este buffer y permiten que entren nuevos elementos del archivo de entrada
 - Cuando el buffer en memoria secundaria se llena → se termina de dormir elementos
 - Las particiones quedan **con más elementos**

- **Clasificación por partes:** Resumen de técnicas de generación de particiones
 - **Interno**
 - Es el más simple
 - Genera particiones más pequeñas que el resto
 - **Selección por reemplazo**
 - Particiones más grandes que el sort interno
 - Tiende a generar muchos registros dormidos
 - **Selección natural**
 - Genera particiones de mayor tamaño que el resto
 - Mayor costo de acceso (generación de archivo de elementos dormidos)

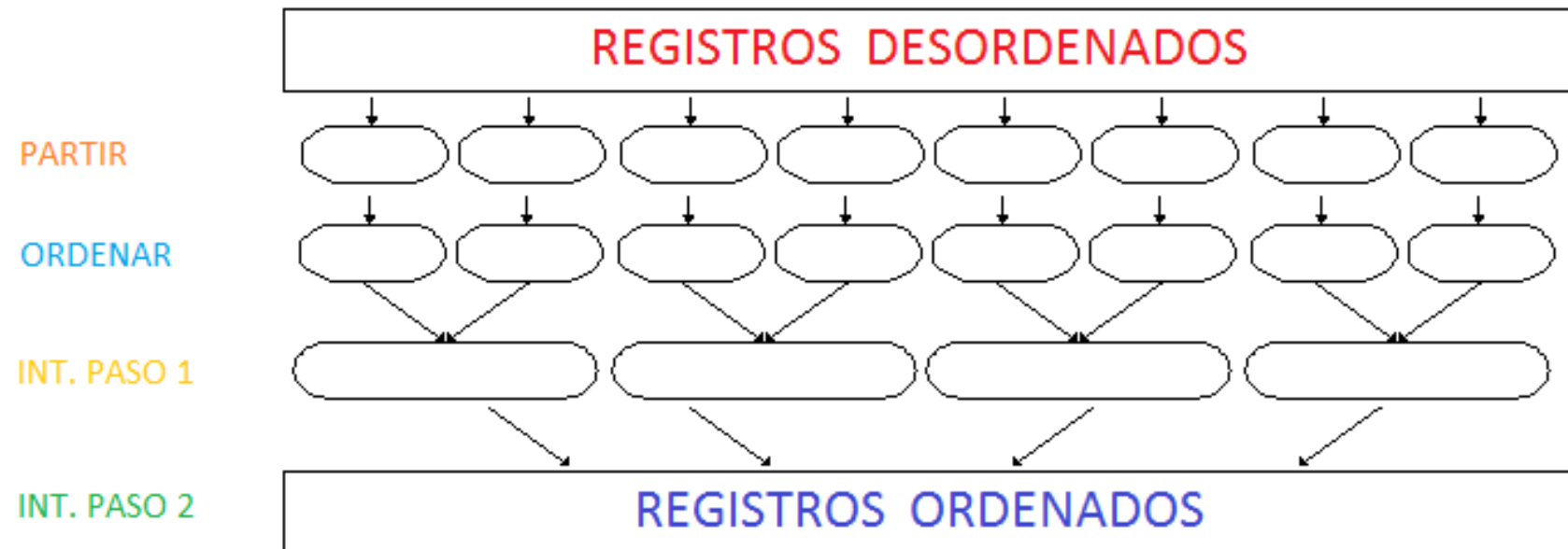
- **Clasificación por partes**

- Una vez generadas las particiones, es posible minimizar la cantidad total de desplazamientos de la clasificación utilizando un **procedimiento de unión más eficiente**
- **Intercalación en más de un paso**
 - Se agregan pasos intermedios que generan **particiones intermedias temporales**
 - Luego se intercalan estos archivos intermedios
 - La **desventaja** es que cada registro se debe leer más de una vez (al formar los archivos intermedios y al formar el archivo ordenado final)

Archivos

Clasificación

- Clasificación por partes: **Int. en más de un paso**



- **Clasificación por partes: Int. en más de un paso**
 - En el ejemplo visto inicialmente, en lugar de intercalar las 40 particiones a la vez, primero se intercala **5 conjuntos de 8 particiones** c/u, y luego se intercala esas **5 particiones intermedias**
 - **1º paso:**
 - Se debe leer datos de las **8** particiones del 1º conjunto simultáneamente. Se podrá leer **1/8** de cada partición
 - Para leer una partición completa se necesita **8 desp.**
 - Como hay 8 particiones $\rightarrow 8 * 8 = \mathbf{64\ desp.}$
 - En total, para los 5 conjuntos $\rightarrow 5 * 64 = \mathbf{320\ desplazamientos}$
 - **2º paso:**
 - Ahora c/u de las 5 particiones es 8 veces más grande $\rightarrow \mathbf{40\ desp.}$
 - Como hay 5 particiones $\rightarrow 5 * 40 = \mathbf{200\ desplazamientos}$
 - **Total: $320 + 200 = \mathbf{520\ desplazamientos}$**

- **Clasificación por partes: Int. en más de un paso**
 - **Conclusiones**
 - Se reduce el número de desplazamientos de **1600** (ejemplo inicial) a **520**
 - Si se cuenta con un archivo del doble de tamaño, es decir, **800.000** registros:
 - Intercalando en **un paso** → **6400** desplazamientos
 - Intercalando en **dos pasos** (10 conjuntos de 8 particiones) → **1440** desplazamientos

Índices

- Es importante **mantener el orden** en un archivo para poder buscar información de forma eficiente → **búsqueda binaria**
- Pero, aunque se utilicen los métodos de **clasificación** e **intercalación** más eficientes:
 - Es muy costoso **ordenar** archivos grandes
 - Es muy costoso **mantener ordenado** los archivos, ya que ante cada cambio se debe reordenar
- Alternativa → **orden lógico**

Índices

- Las búsquedas secuencial y binaria **no son naturales**, no se usan en la vida cotidiana
 - Al buscar un tema en un libro → se utiliza su **índice**, y luego se accede a la página rápidamente
 - En una biblioteca se tienen diferentes criterios de búsqueda de libros → se dispone de un **conjunto de índices** (autor, tema, título, etc.) que permite encontrar el libro deseado

Índices

Estructura de datos

- El índice es una **estructura auxiliar** al archivo que nos permite acceder a sus datos
 - Es una **alternativa eficiente** al momento de buscar información
 - Varios índices nos pueden proporcionar **diferentes caminos de acceso** a un mismo elemento de un archivo
 - Permite **imponer orden en un archivo sin que éste realmente se reacomode**

Índices

Estructura de datos

- Archivo adicional **ordenado** con registros de **longitud fija**, independientemente de la estructura de datos del archivo original
 - Solo contiene un par de datos → **mucho más pequeño**
 - Según la estructura del archivo de datos:
 - Longitud fija: (**clave**, **NRR**)
 - Longitud variable: (**clave**, **distancia en bytes**)

- Ahora se debe mantener este nuevo archivo índice, además del archivo de datos original
 - El **orden** sólo se debe mantener sobre el archivo índice
→ **mucho más pequeño** que el archivo de datos
 - La **búsqueda de un dato** se realiza:
 1. En el **archivo índice** → se obtiene la **referencia** al registro del archivo de datos
 2. Usando esa referencia → se accede directamente a la información buscada en el **archivo de datos**

Índices

- Ej: **archivo de datos** con reg. de **long. variable**

| Dir. Reg. | Cía | IdCia | Título | Compositores | Artista |
|-----------|-----|-------|----------------------|----------------|--------------------|
| 32 | UNI | 2312 | Paranoid Android | Radiohead | Radiohead |
| 77 | SON | 2626 | If I Fell | John Lennon | The Beatles |
| 132 | WAR | 23699 | The Scientist | Coldplay | Coldplay |
| 167 | UNI | 3795 | Jealous Guy | John Lennon | John Lennon |
| 211 | BMG | 38358 | Color Esperanza | Coti | Diego Torres |
| 256 | EMI | 18807 | Ticket To Ride | John Lennon | The Beatles |
| 300 | SON | 75016 | Baja a la Tierra | Kevin Johansen | Kevin J.+ The Nada |
| 353 | BMG | 31809 | Samson | Regina Spektor | Regina Spektor |
| 396 | EMI | 13920 | All You Need Is Love | John Lennon | Paul McCartney |
| 422 | SON | 245 | Grita | Jarabe de Palo | Jarabe de Palo |

Índices

Índice primario

- Definición de la estructura del **archivo de índice primario**
 - Debe contener **registros de longitud fija** y mantenerse ordenado para permitir realizar una **búsqueda binaria** sobre sus datos
 - El **índice primario** es el que se basa en la clave primaria del archivo de datos
 - Se supone que en el ejemplo la **clave primaria** está dada por los campos **Cía + IdCía** → **UNÍVOCA**
 - Forma canónica: **Cía en mayúsculas + IdCía** → Ej: **UNI2312**

- Definición de la estructura del **archivo de índice primario**
 - **Campo de clave: 12 caracteres**, alineados a izquierda y completado con blancos
 - **Campo de referencia: nro entero**, que es la dirección del primer byte del registro correspondiente en el archivo de datos
 - **Tamaño pequeño** → es muy probable que sea posible trabajar en memoria
 - La **búsqueda binaria** será **más rápida** que la realizada sobre un archivo de datos clasificado

Índices

Índice primario

- En el ejemplo dado

| Clave | Ref |
|----------|-----|
| BMG31809 | 353 |
| BMG38358 | 211 |
| EMI13920 | 396 |
| EMI18807 | 256 |
| SON245 | 422 |
| SON2626 | 77 |
| SON75016 | 300 |
| UNI2312 | 32 |
| UNI3795 | 167 |
| WAR23699 | 132 |

| Dir. Reg. | Registro de datos |
|-----------|--|
| 32 | UNI 2312 Paranoid Android Radiohead Radiohead |
| 77 | SON 2626 If I Fell John Lennon The Beatles |
| 132 | WAR 23699 The Scientist Coldplay Coldplay |
| 167 | UNI 3795 Jealous Guy John Lennon John Lennon |
| 211 | BMG 38358 Color Esperanza Coti Diego Torres |
| 256 | EMI 18807 Ticket To Ride John Lennon The Beatles |
| 300 | SON 75016 Baja a la Tierra Kevin Johansen Kevin J.+ The Nada |
| 353 | BMG 31809 Samson Regina Spektor Regina Spektor |
| 396 | EMI 13920 All You Need Is Love John Lennon Paul McCartney |
| 422 | SON 245 Grita Jarabe de Palo Jarabe de Palo |

Índices

Índice primario

- Búsqueda de un registro (por clave primaria)
 - Se desea encontrar el registro correspondiente al título “Grita” → su clave primaria es **SON245**
 - Utilizando esa clave se realiza una búsqueda binaria en el archivo índice y se obtiene la referencia **422**
 - Ahora sí, se puede acceder directamente al **byte 422** del archivo de datos y **recuperar el registro buscado**

- Al utilizar una estructura adicional como lo es el archivo índice:
 - Las **operaciones básicas** sobre el archivo de datos cambiarán y/o se le sumarán algunas operaciones sobre el archivo índice asociado
- Operaciones básicas a analizar
 - Creación
 - Carga en memoria
 - Agregar elementos
 - Modificar elementos
 - Eliminar elementos

Índices

Operaciones básicas

- Creación
 - Al **iniciar un nuevo archivo**, se debe crear tanto el archivo de **datos** como el archivo **índice**
 - Ambos se crean **vacíos** (sólo con el registro cabecera)
- Carga en memoria
 - Se supone que el archivo índice es lo **suficientemente pequeño** como para caber en memoria principal
 - Se almacena en un arreglo

- Agregar un elemento
 - Implica agregar registros en ambos archivos
 - **Datos**
 - Se agrega el registro al final del archivo
 - Se guarda el **NRR** (long.fija) o la **distancia en bytes** (long.var) para usarlo posteriormente en el archivo índice
 - ¿Se debe reordenar el archivo?
 - **Índice**
 - Usando el valor guardado al agregar el registro en el archivo de datos, se arma el registro (**clave, NRR**) ó (**clave, distBytes**)
 - Se agrega el registro al final del archivo
 - ¿Se debe reordenar el archivo?

- Modificar un elemento
 - Se cambia la clave primaria = (eliminar + agregar)
 - No se cambia la clave primaria
 - **Longitud Fija**
 - El índice **no se altera** → el **NRR** se mantiene constante
 - **Longitud Variable**
 - Si el registro **no aumenta su tamaño**, **no se altera** → mantendrá su posición en el archivo de datos
 - Si el registro **ocupa más** (ya no entra en el mismo lugar) → **cambiará su posición** en el archivo de datos y por lo tanto, el índice también **deberá ser actualizado**

- Eliminar un elemento
 - **Archivo de datos**
 - Baja lógica
 - Se utiliza alguna de las técnicas vistas para reutilizar el espacio
 - **Archivo índice**
 - Baja física o lógica
 - ¿Reutilización de espacio?

Índices

Índices secundarios

- ¿Cómo se obtienen las claves para realizar las búsquedas?
 - En el ej. visto, se realizó la búsqueda del registro con el título “**Grita**” usando su clave (**SON245**)

| Clave | Ref |
|----------|-----|
| BMG31809 | 353 |
| BMG38358 | 211 |
| EMI13920 | 396 |
| EMI18807 | 256 |
| SON245 | 422 |
| SON2626 | 77 |
| SON75016 | 300 |
| UNI2312 | 32 |
| UNI3795 | 167 |
| WAR23699 | 132 |

| Dir. Reg. | Registro de datos |
|-----------|--|
| 32 | UNI 2312 Paranoid Android Radiohead Radiohead |
| 77 | SON 2626 If I Fell John Lennon The Beatles |
| 132 | WAR 23699 The Scientist Coldplay Coldplay |
| 167 | UNI 3795 Jealous Guy John Lennon John Lennon |
| 211 | BMG 38358 Color Esperanza Coti Diego Torres |
| 256 | EMI 18807 Ticket To Ride John Lennon The Beatles |
| 300 | SON 75016 Baja a la Tierra Kevin Johansen Kevin J.+ The Nada |
| 353 | BMG 31809 Samson Regina Spektor Regina Spektor |
| 396 | EMI 13920 All You Need Is Love John Lennon Paul McCartney |
| 422 | SON 245 Grita Jarabe de Palo Jarabe de Palo |

Índices

Índices secundarios

- En algunos casos no resulta natural solicitar un dato por clave primaria
- En su lugar se utiliza un **campo más fácil de recordar**
 - En el ejemplo, la búsqueda podría ser directamente por el título “**Grita**”
 - Para poder realizar esa búsqueda, se necesita definir un **índice secundario** por dicho campo

Índices

Índices secundarios

- Un índice secundario **relaciona una clave secundaria con la clave primaria** correspondiente
- Las claves secundarias **pueden repetirse**
- Para buscar un registro por clave secundaria:
 - Se accede por **clave secundaria** al índice secundario → se obtiene la **clave primaria**
 - Se accede por **clave primaria** al índice primario → se obtiene la **referencia al archivo de datos**
 - Utilizando esta **referencia** se accede en forma directa al registro en el archivo de datos

Índices

Índices secundarios

- Ej: ya se contaba con dos archivos: **índice primario** y **datos**

| Clave | Ref |
|----------|-----|
| BMG31809 | 353 |
| BMG38358 | 211 |
| EMI13920 | 396 |
| EMI18807 | 256 |
| SON245 | 422 |
| SON2626 | 77 |
| SON75016 | 300 |
| UNI2312 | 32 |
| UNI3795 | 167 |
| WAR23699 | 132 |

| Dir. Reg. | Registro de datos |
|-----------|--|
| 32 | UNI 2312 Paranoid Android Radiohead Radiohead |
| 77 | SON 2626 If I Fell John Lennon The Beatles |
| 132 | WAR 23699 The Scientist Coldplay Coldplay |
| 167 | UNI 3795 Jealous Guy John Lennon John Lennon |
| 211 | BMG 38358 Color Esperanza Coti Diego Torres |
| 256 | EMI 18807 Ticket To Ride John Lennon The Beatles |
| 300 | SON 75016 Baja a la Tierra Kevin Johansen Kevin J.+ The Nada |
| 353 | BMG 31809 Samson Regina Spektor Regina Spektor |
| 396 | EMI 13920 All You Need Is Love John Lennon Paul McCartney |
| 422 | SON 245 Grita Jarabe de Palo Jarabe de Palo |

Índices

Índices secundarios

- Ahora, por ejemplo, se puede añadir un tercer archivo → un **índice secundario** por el campo **compositor**

| INDICE DE COMPOSITORES | |
|------------------------|----------------|
| Clave Secundaria | Clave Primaria |
| Coldplay | WAR23699 |
| Coti | BMG38358 |
| Jarabe de Palo | SON245 |
| John Lennon | EMI13920 |
| John Lennon | EMI18807 |
| John Lennon | SON2626 |
| John Lennon | UNI3795 |
| Kevin Johansen | SON75016 |
| Radiohead | UNI2312 |
| Regina Spektor | BMG31809 |

Índices

Índices secundarios

- Si se busca el registro correspondiente al compositor “Kevin Johansen”
 - Se accede al **índice secundario** y se busca la entrada “Kevin Johansen” → se obtiene la clave primaria **SON75016**
 - Se accede al **índice primario** y se busca la entrada con la clave primaria conseguida → se obtiene la referencia **300**, que nos dirigirá directamente al registro buscado en el **archivo de datos**

Índices

Índices secundarios

- Si se busca el registro correspondiente al compositor “John Lennon”
 - Se accede al **índice secundario** y se busca el registro “John Lennon” → se obtiene **más de una clave primaria**, ya que varias entradas tienen ese compositor
 - Cuando hay varios registros con una misma clave secundaria → existe un **segundo criterio de ordenación**

Índices

Operaciones básicas (Índ. Sec.)

- Creación
 - Al iniciar un nuevo archivo, se debe crear el archivo de datos, el archivo de índice primario y **todos los archivos de índices secundarios necesarios**
 - Habrá un archivo de índice secundario por cada criterio de búsqueda que se desee utilizar
 - Todos los archivos se crean **vacíos** (sólo con el registro cabecera)

Índices

Operaciones básicas (Índ. Sec.)

- Agregar un elemento
 - Toda alta en el archivo de datos genera una inserción en el índice secundario → **reacomodación del archivo de índice secundario**
 - La **claves duplicadas** implican un **reordenamiento de segundo nivel** de acuerdo a la clave primaria que referencian
 - Bajo costo si el índice está en memoria principal

Índices

Operaciones básicas (Índ. Sec.)

- Modificar un elemento
 - Se cambia la clave primaria = (eliminar + agregar)
 - No se cambia la clave primaria
 - Se cambia la clave secundaria → reacomodación del archivo de índice secundario
 - No se cambia la clave secundaria → no hay cambios

Índices

Operaciones básicas (Índ. Sec.)

- Eliminar un elemento
 - Al eliminar un registro del archivo de datos se debe:
 - Eliminar **referencia en índice primario**
 - Eliminar **referencias en índices secundarios**
 - Puede ser algo costoso
 - ¿Alternativas?

Índices

Operaciones básicas (Índ. Sec.)

- Eliminar un elemento (alternativa)
 - Sólo borrar la referencia del índice primario → servirá como **barrera de protección**
 - **Beneficio**
 - La barrera aísla a los índices secundarios de los cambios en el archivo de datos
 - No hay un reacomodo por cada eliminación
 - **Costo**
 - Se sigue ocupando el espacio → puede ser un problema si el archivo es muy volátil
 - Es posible programar borrados físicos de los índices secundarios

Índices

Índices secundarios

- Problema: la **repetición de información**
 - Un archivo de índices secundarios **se debe reacomodar** con cada inserción
 - Incluso si la clave secundaria ya existe, dado que existe un segundo orden por clave primaria
 - Misma clave con **varias ocurrencias** en distintos registros
 - Se desperdicia espacio
 - Menor probabilidad de que el índice quepa en memoria

Índices

Índices secundarios

- La repetición de información → [Solución 1](#)
- **Arreglo**: clave + vector de punteros con ocurrencias

| | | | | |
|-------------|----------|----------|---------|---------|
| John Lennon | EMI13920 | EMI18807 | SON2626 | UNI3795 |
|-------------|----------|----------|---------|---------|

- Al agregar un nuevo registro con una **clave nueva** → se genera un arreglo con la clave y un elemento en el vector de punteros
- Al agregar un nuevo registro de una **clave existente** → no se debe reacomodar el índice, **sólo se debe reacomodar el vector de ocurrencias**
- **Problema** → elección del tamaño (**fijo**) del vector
 - Posibles casos en que sea **insuficiente**
 - Posibles casos en que **sobre espacio** → fragmentación interna

Índices

Índices secundarios

- La repetición de información → **Solución 2**
 - **Listas invertidas**: archivos en los que una clave sec. lleva a un conjunto de una o más claves primarias → **lista de referencias de claves primarias**
 - **No hay restricciones** en cuanto a la cantidad de claves que es posible guardar
 - **No se pierde espacio** → no hay reserva
 - Al agregar un elemento a la lista **no es necesario** realizar una **reorganización completa**
 - Se organiza físicamente con un **nuevo archivo** al que se referencia desde el archivo de índice secundario

Índices

Índices secundarios

- Las listas deben **mantener el orden localmente**

| INDICE DE COMPOSITORES | | |
|------------------------|-----------------------------|------|
| NRR | Clave Secundaria | Link |
| 0 | Coldplay | 2 |
| 1 | Coti | 4 |
| 2 | Jarabe de Palo | 9 |
| 3 | John Lennon | 8 |
| 4 | Kevin Johansen | 6 |
| 5 | Radiohead | 0 |
| 6 | Regina Spektor | 7 |

| LISTA DE CLAVES PRIMARIAS | | |
|---------------------------|--------------------------|------|
| NRR | Clave Primaria | Link |
| 0 | UNI2312 | -1 |
| 1 | SON2626 | 3 |
| 2 | WAR23699 | -1 |
| 3 | UNI3795 | -1 |
| 4 | BMG38358 | -1 |
| 5 | EMI18807 | 1 |
| 6 | SON75016 | -1 |
| 7 | BMG31809 | -1 |
| 8 | EMI13920 | 5 |
| 9 | SON245 | -1 |

Índices

Índices secundarios

- La repetición de información → Solución 2
 - Ventajas
 - Índice secundario más pequeño
 - Agregar/modificar una clave secundaria → único reacomodamiento (y menos costoso) sobre índice sec.
 - Agregar/eliminar claves primarias para una clave secundaria ya existente → sólo se debe modificar el archivo de listas (manteniendo el orden localmente)

Índices

Índices secundarios

- La repetición de información → Solución 2
 - Desventajas
 - Es conveniente que el **archivo de listas** esté en **memoria principal** sino se podrían generar muchos desplazamientos en disco
 - Por cada índice secundario se precisa dos archivos: el **archivo del índice secundario** y el **archivo de listas**
 - Costoso si hay muchos índices secundarios

Índices

Índices selectivos

- Los **índices selectivos** contienen claves sólo para una **porción de los registros** del archivo de datos
 - A partir del índice definido → **sólo es visible un subconjunto** de los registros del archivo
 - El índice actúa como **filtro**
 - En el ejemplo, se podría definir un índice selectivo que permita ver sólo los registros que pertenezcan a un **género musical en particular**

- **Ventajas** del uso de índices
 - Archivos índices → **registro de longitud fija**
 - Permite búsqueda binaria aún cuando el archivo de datos contenga registros de longitud variable
 - **Menor tamaño** que el archivo de datos
 - Mayor probabilidad de estar en memoria principal
 - La búsqueda binaria es más eficiente
 - El mantenimiento es menos costoso

- **Desventaja** del uso de índices
 - Existe un **procesamiento adicional**. Debido a la **reacomodación** de los índices ante altas o bajas en los archivos de datos
- Existen alternativas para mejorar aún más la eficiencia en el acceso a los datos
 - Árboles balanceados
 - Dispersión de archivos