

File Organization

Métodos de Organización

Semana 02



Tipos de búsqueda

1. Devolver todos los registros

- *Record[] scanALL()* ✓

2. Devolver registro dada su posición lógica

- *Record search(int n)* ✓

3. Devolver registro(s) dado el valor de alguno de sus campos (search-key)

- *Record[] search(<FieldType> key)* ?

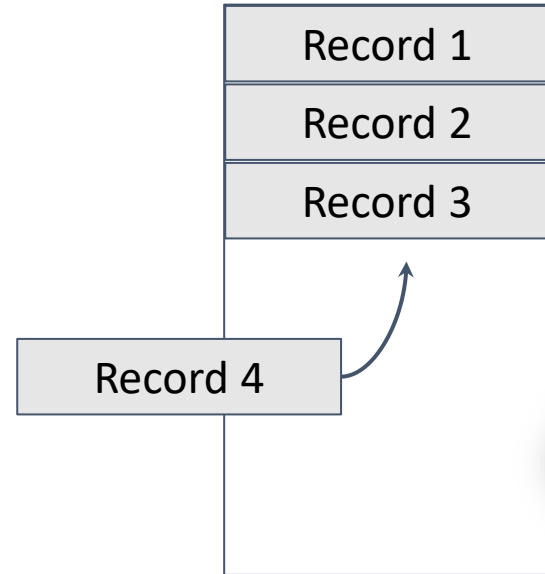
Métodos de Organización de Archivos

El método de organización de archivos depende de los medios de almacenamiento, tamaño de archivo, tipo de acceso, grado de actividad y volatilidad.

1. **Heap Files**
2. **Sequential Files**
3. **Random Files**
4. **Indexed Sequential Files**

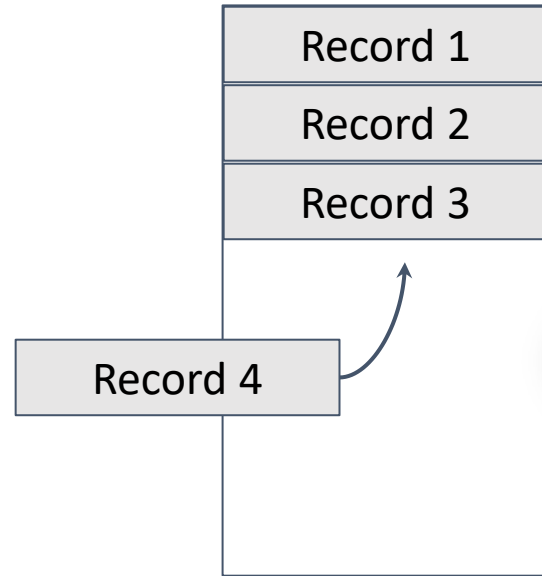
Heap File Organization

- En un Heap File los registros son almacenados en **ubicaciones adyacentes**, conforme van llegando.
- No mantiene un orden físico respecto a campo.
- Se puede manejar registros de longitud variable.
- Son gestionados de alguna manera por el sistema operativo.



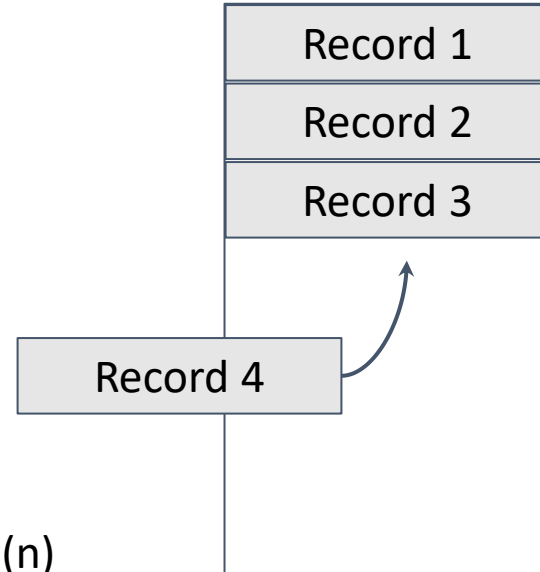
Heap File Organization

- Usos:
 - Archivos transaccionales. En donde los registros se almacenan en orden cronológico y se recuperan en orden inverso.
- Generalmente son usados junto con otras estructuras de acceso.



Heap File Organization

- Ventajas
 - Las inserciones son fáciles y muy eficientes. $O(1)$
- Desventajas
 - Acceder a un registro por un atributo dado requiere revisar el archivo completo (muy costoso). $O(n)$



Sequential [Ordered] File Organization

- El objetivo es poder aplicar la búsqueda binaria para conseguir una complejidad de acceso a memoria secundaria $O(\log n)$
- Para ello, el archivo debe mantener los registros ordenados físicamente en base al valor del campo de búsqueda (key).
- Principalmente se ordena en base a la llave primaria de la tabla.

datos.dat

Id	<u>Nombre</u>	Ciclo
P-102	Andrea	5
P-250	Carlos	7
P-362	Cinthya	3
P-231	Josimar	5
P-087	Jorge	1
P-312	Mabel	3
P-982	Saulo	9

Sequential [Ordered] File Organization

Búsqueda binaria en el archivo:

- El algoritmo de búsqueda binaria es usado para localizar un registro en el archivo dado un valor de búsqueda k .
- Se requiere **$O(\log N)$** accesos a memoria secundaria.
- En la búsqueda se debe descartar los registros marcados como eliminados.

```
Require:  $k$ 
 $l = 0$ 
 $u = \text{size}() - 1$ 
while  $u \geq l$  do
     $m = \lfloor (l + u) / 2 \rfloor$ 
     $re = \text{readRecord}(m)$ 
    if  $re.key < k$  then
         $u = m - 1$ 
    else if  $re.key > k$  then
         $l = m + 1$ 
    else
        return  $m$ 
    end if
end while
```


Sequential [Ordered] File Organization

¿Cómo mantener el archivo ordenado?

¿Cuánto cuesta insertar?

¿Cuánto cuesta eliminar?

datos.dat

Id	<u>Nombre</u>	Ciclo
P-102	Andrea	5
P-250	Carlos	7
P-362	Cintha	3
P-231	Josimar	5
P-087	Jorge	1
P-312	Mabel	3
P-982	Saulo	9

Sequential File Organization

Estrategia del espacio auxiliar:

- Las nuevas inserciones se van almacenando en un espacio auxiliar
- Mantener un limite máximo de K registros en el espacio auxiliar
- La búsqueda se debe hacer en ambos espacios.
- Cada cierto tiempo el archivo de datos debe reconstruirse con los registros del espacio auxiliar.

datos.dat

Id	<u>Nombre</u>	Ciclo
P-102	Andrea	5
P-250	Carlos	7
P-362	Cinthy	3
P-231	Josimar	5
P-087	Jorge	1
P-312	Mabel	3
P-982	Saulo	9

aux.dat

P-312	Gabriel	3
P-123	Diana	6

...

Sequential File Organization

Inserciones enlazadas:

- Localizar la posición en donde será insertado el nuevo registro.
 - Si el espacio está libre, insertar.
 - Sino, insertar el registro en un espacio auxiliar.
 - En este caso, los punteros deberían ser actualizados.
- Se requiere reorganizar el archivo original cada cierto tiempo mezclando ordenadamente con el espacio auxiliar.

datos.dat

Id	<u>Nombre</u>	Ciclo	
P-102	Andrea	5	
P-250	Carlos	7	
P-362	Cinthy	3	
P-231	Josimar	5	
P-087	Jorge	1	
P-312	Mabel	3	
P-982	Saulo	9	


aux.dat

P-312	Gabriel	3	
P-123	Diana	6	

...

datos.dat

	Id	<u>Nombre</u>	Ciclo	1(d)
1	P-102	Andrea	5	2(d)
2	P-250	Carlos	7	3(d)
3	P-362	Cinthya	3	4(d)
4	P-231	Josimar	5	5 (d)
5	P-087	Jorge	2	6 (d)
6	P-312	Mabel	3	7(d)
7	P-982	Saulo	9	-1 (d)



Insertar

P-088	Gabriel	4	
P-312	Gonzalo	3	
P-087	Maria	2	
P-014	Abel	5	

aux.dat


1
2
3
4

Sequential File Organization

Eliminación de un registro:

- Se utiliza los punteros para saltar las tuplas eliminadas.
- En la reconstrucción del archivo se serán completamente eliminados.

Id	<u>Nombre</u>	Ciclo	
P-102	Andrea	5	
P-250	Carlos	7	
P-362	Cinthya	3	
P-231	Josimar	5	-1
P-087	Jorge	1	
P-312	Mabel	3	-1
P-982	Saulo	9	



Eliminar Jorge

datos.dat

	Id	<u>Nombre</u>	Ciclo	4(a)
1	P-102	Andrea	5	2(d)
2	P-250	Carlos	7	3(d)
3	P-362	Cinthya	3	1 (a)
4	P-231	Josimar	5	5 (d)
5	P-087	Jorge	2	6 (d)
6	P-312	Mabel	3	3(a)
7	P-982	Saulo	9	-1 (d)

Eliminar: Josimar, Jorge

→ Antes de eliminar

aux.dat

1	P-088	Gabriel	4	2(a)
2	P-312	Gonzalo	3	4(d)
3	P-087	Maria	2	7(d)
4	P-014	Abel	5	1(d)

datos.dat

	Id	<u>Nombre</u>	Ciclo	4(a)
1	P-102	Andrea	5	2(d)
2	P-250	Carlos	7	3(d)
3	P-362	Cinthya	3	1(a)
4	P-231	Josimar	5	0
5	P-087	Jorge	2	0
6	P-312	Mabel	3	3(a)
7	P-982	Saulo	9	-1 (d)

Eliminar: Josimar, Jorge

→ Después de eliminar

aux.dat

1	P-088	Gabriel	4	2 (a)
2	P-312	Gonzalo	3	6(d)
3	P-087	Maria	2	7(d)
4	P-014	Abel	5	1(d)

datos.dat

	Id	<u>Nombre</u>	Ciclo	1(d)
1	P-102	Andrea	5	2(d)
2	P-250	Carlos	7	3(d)
3	P-362	Cinthy	3	2 (a)
4	P-231	Josimar	5	0
5	P-087	Jorge	2	0
6	P-312	Mabel	3	3 (a)
7	P-982	Saulo	9	0 (d)

aux.dat

1	P-088	Gonzalo	4	6 (d)
2	P-312	Gabriel	3	1 (a)
3	P-087	Maria	2	7 (d)

Reconstrucción

Id	<u>Nombre</u>	Ciclo	
P-102	Andrea	5	
P-250	Carlos	7	
P-362	Cinthy	3	
P-312	Gabriel	3	
P-088	Gonzalo	4	
P-312	Mabel	3	
P-087	Maria	2	
P-982	Saulo	9	

Insertar ordenadamente
siguiendo los punteros

datos.dat

ordenado	7				
	1	P-102	Andrea	5	2
	2	P-250	Carlos	7	3
	3	P-362	Cinthy	3	9
	4	P-231	Josimar	5	9
	5	P-087	Jorge	2	9
	6	P-312	Mabel	3	10
no ordenado	7	P-982	Saulo	9	-1
	8	P-088	Gonzalo	4	6
	9	P-312	Gabriel	3	8
	10	P-087	Maria	2	7

Reconstrucción

Id	<u>Nombre</u>	Ciclo	
P-102	Andrea	5	2
P-250	Carlos	7	3
P-362	Cinthy	3	4
P-312	Gabriel	3	5
P-088	Gonzalo	4	6
P-312	Mabel	3	7
P-087	Maria	2	8
P-982	Saulo	9	-1

Insertar ordenadamente
siguiendo los punteros

Se puede mantener el espacio
auxiliar en el mismo archivo de datos

Sequential File Organization

datos.dat

Id	<u>Nombre</u>	Ciclo
P-102	Andrea	5
P-250	Carlos	7
P-362	Cinthya	3
P-231	Josimar	5
P-087	Jorge	1
P-312	Mabel	3
P-982	Saulo	9

aux.dat

P-312	Gabriel	3
P-123	Diana	6

Dos formas de manejar
el espacio auxiliar

- Search:
- Insert:
- Rebuild:
- Remove:

datos.dat

Id	<u>Nombre</u>	Ciclo	
P-102	Andrea	5	
P-250	Carlos	7	
P-362	Cinthya	3	
P-231	Josimar	5	
P-087	Jorge	1	
P-312	Mabel	3	
P-982	Saulo	9	

aux.dat

P-312	Gabriel	3	
P-123	Diana	6	




- Search:
- Insert:
- Rebuild:
- Remove:

Sequential File Organization

- Ventajas
 - Búsquedas eficientes
- Desventajas
 - Son difíciles de mantener.
 - Inserciones y eliminaciones requieren de una localización previa del registro.
 - Costo extra para reorganizar el archivo.

Id	<u>Nombre</u>	Ciclo	
P-102	Andrea	5	
P-250	Carlos	7	
P-362	Cinthya	3	
P-231	Josimar	5	
P-087	Jorge	1	
P-312	Mabel	3	
P-982	Saulo	9	



AVL File

- ¿Y si mantenemos los registros ordenados usando la estrategia de Árbol Binario de Búsqueda?
 - Lo llamaremos BST File / AVL File
 - ¿Cómo cambia el archivo?
 - ¿Complejidad de inserción, eliminación y búsqueda?

AVL File

	<u>Cod</u>	Nombre	Ciclo	Left	Right
1	P-271	Josimar	5	2	3
2	P-255	Manuel	8	4	-1
3	P-362	Cinthya	3	-1	5
4	P-224	Andrea	2	-1	-1
5	P-887	Benjamin	9	-1	-1

Random [Direct] File Organization

- Los registros en el archivo no necesitan estar físicamente ordenados.
- Se usa un diccionario para mantener la relación entre el key y la ubicación del registro
- Cuando necesitamos acceder al registro, el key es usado para encontrar su dirección.
- **Hashing** es una de las técnicas usadas para el mapeo de registros y direcciones.

index.dat

key	Address
Josimar	R1
Manuel	R2
Cinthya	R3
Andrea	R4
Benjamin	R5

datos.dat

	Id	<u>Nombre</u>	Ciclo
R1	P-231	Josimar	5
R2	P-255	Manuel	8
R3	P-362	Cinthya	3
R4	P-231	Andrea	2
R5	P-887	Benjamin	9

Random File Organization

- **Ventajas**

- Inserciones eficientes
- Búsqueda eficiente
 - Si se procesa en memoria RAM.
 - Si se mantiene el diccionario ordenado.

RAM

- **Desventajas**

- Espacio extra para mantener el diccionario.
- Estrategias de organización si se producen **colisiones** en el search-key.
 - Hash Techniques

key	Address
Andrea	R4
Benjamin	R5
Cinthya	R3
Josimar	R1
Manuel	R2

Id	<u>Nombre</u>	Ciclo	
P-231	Josimar	5	R1
P-255	Manuel	8	R2
P-362	Cinthya	3	R3
P-231	Andrea	2	R4
P-887	Benjamin	9	R5

index.dat

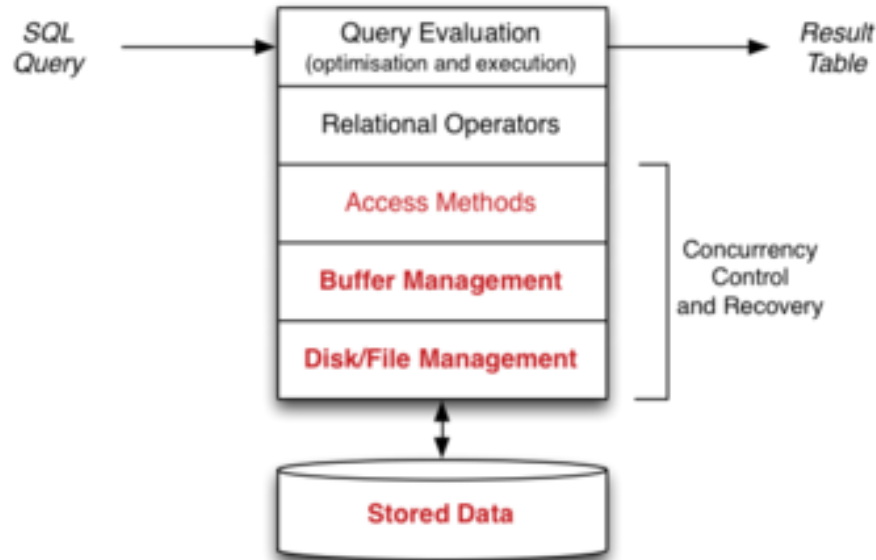
key	Address
Andrea	R4
Manuel	R2
Cinthya	R3
Josimar	R1
Benjamin	R5
Gabriel	R6

datos.dat

Id	<u>Nombre</u>	Ciclo	
P-231	Josimar	5	R1
P-255	Manuel	8	R2
P-362	Cinthya	3	R3
P-231	Andrea	2	R4
P-887	Benjamin	9	R5
P-231	Gabriel	2	R6

Laboratorio 2.1

Empaquetando Registros en Bloques



Empaquetando Registros en Bloques

Se presentan dos casos:

1. La longitud del registro es *menor* que el tamaño del bloque

- Este es el caso más común
- El máximo número de registros que son almacenados dentro de un bloque es llamado “**blocking factor**”: $b = \lfloor B/r \rfloor$, en donde:
 - **B** es el tamaño del bloque en bytes
 - **r** la longitud del registro en bytes

```
char buffer[1024];  
read(&buffer, 1024);  
vector<Registro> desempaquetar(buffer, 1024)
```

Empaquetando Registros en Bloques

Se presentan dos casos:

2. La longitud del registro es **mayor** que el tamaño del bloque

- Se utiliza la organización extendida (spanned organization).
- Un registro extendido puede abarcar diferentes bloques.



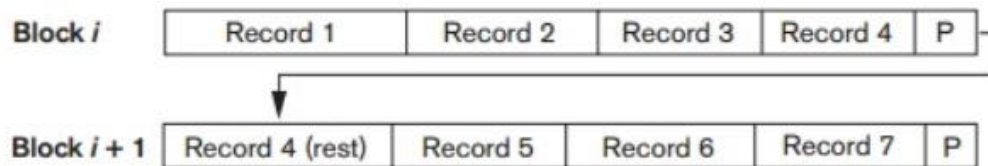
- File Blocks: secuencia de bloques que contienen todos los registros del archivo.

Empaquetando Registros en Bloques

Se presentan dos casos:

2. La longitud del registro es **mayor** que el tamaño del bloque

- También se utiliza organización extendida para evitar espacios libres en los buffers.



Leer más ...

*File Structures: an object oriented approach with C++. Michael Folk. **Chapter 4.***

Indexed Sequential Access Method ISAM

Indexed Sequential Access Method

Combina las ventajas de ambos métodos: acceso secuencial y random.

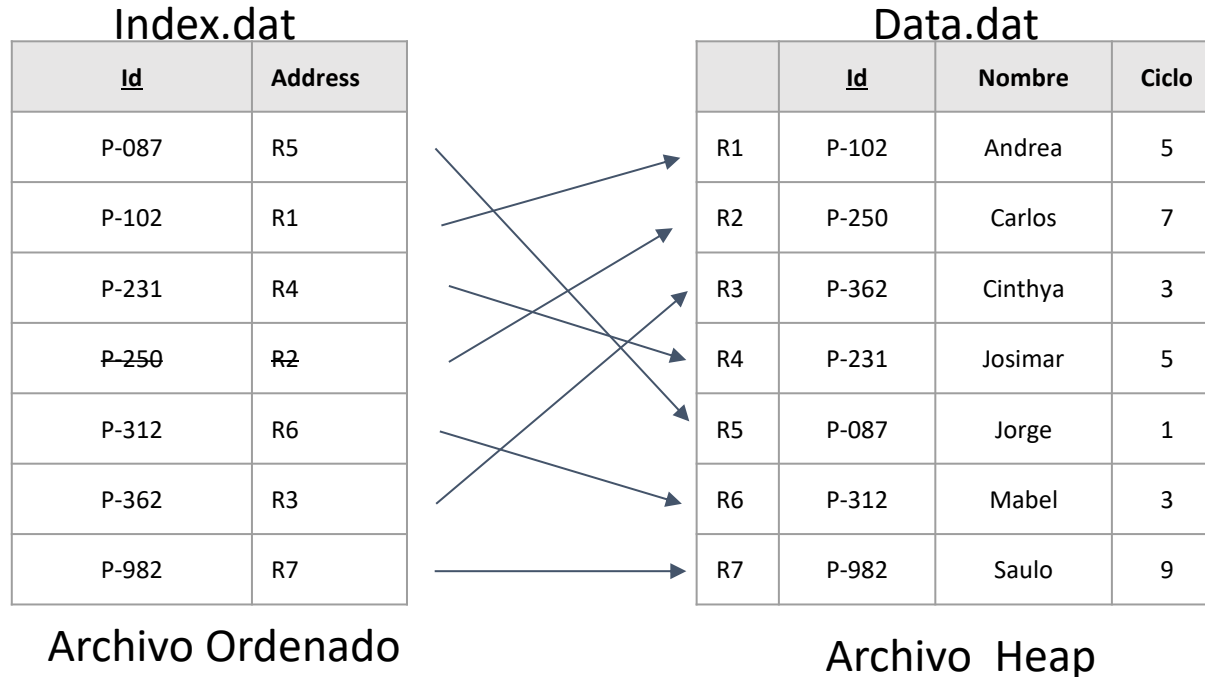
Generalmente se estructura de la siguiente manera:

- ***Index File:*** Aquí se guarda el **key** con los punteros a los respectivos registros [páginas] en el archivo de datos. **Archivo ordenado** [necesario].
- ***Data File:*** *Mantiene los datos originales.*
 - *Opcionalmente se puede mantener en un **archivo semi-ordenado**.*

Indexed Sequential Access Method

Dense Index-file:

- Contiene una entrada por cada valor del search-key del Data-File.



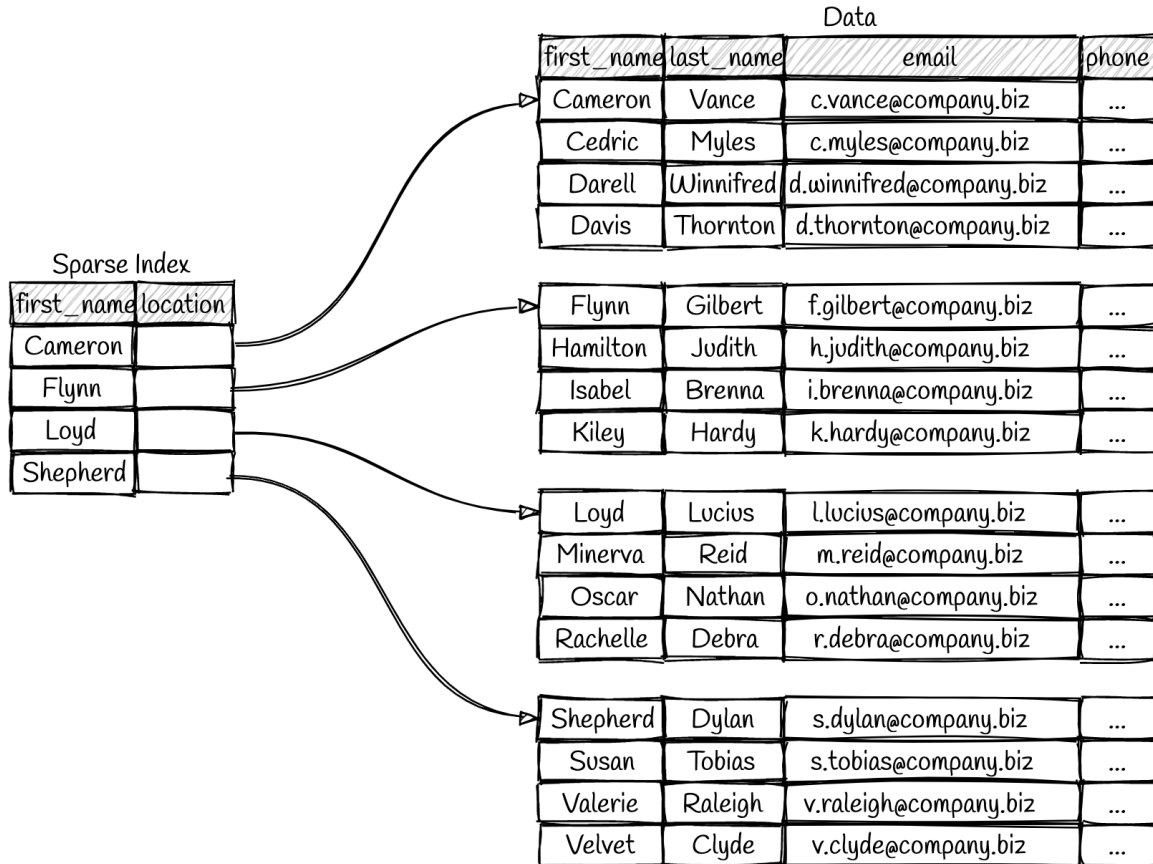
Indexed Sequential Access Method

Sparse Index-file:

- Una entrada puede asociar varios registros en el Data-File.
- Generalmente se aplica cuando los datos ya están ordenados.

¿Ventajas respecto al Sequential File?

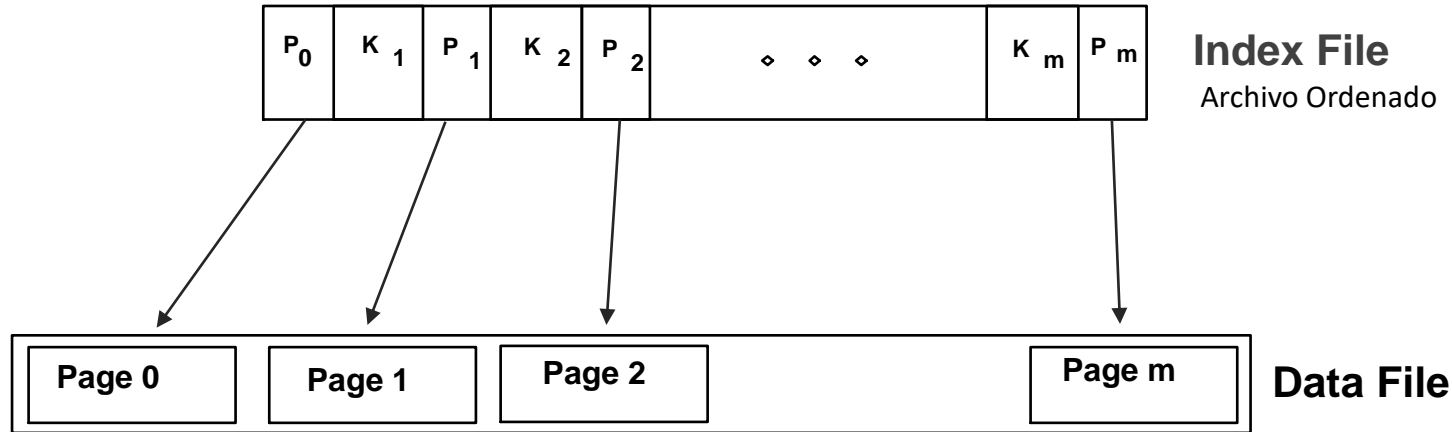
¿Como se construye el índice?



Indexed Sequential Access Method

Sparse Index-file:

- Una entrada puede asociar varios registros en el Data-File.
- Generalmente se aplica cuando los datos ya están ordenados.

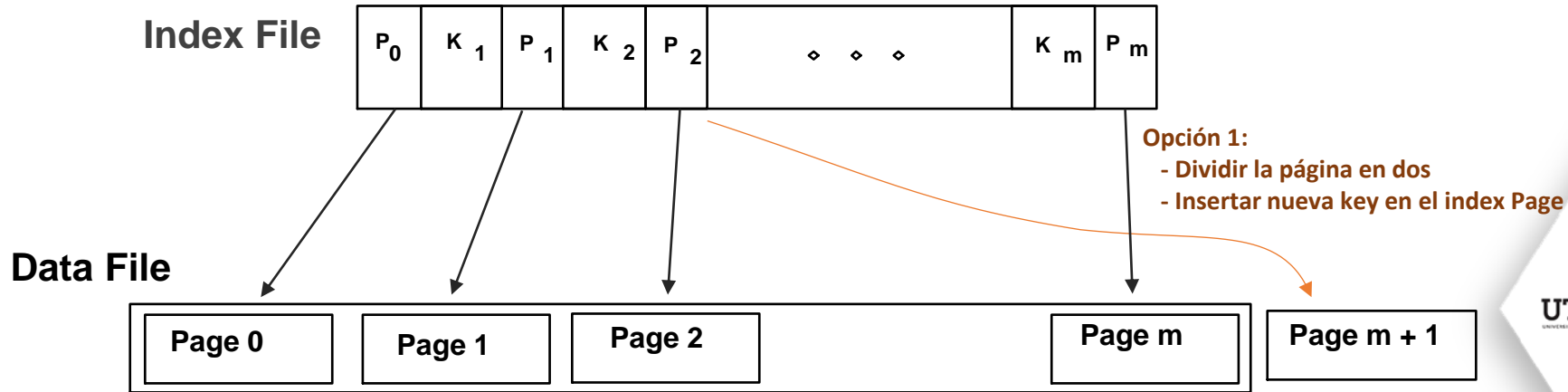


¿Cómo insertar un nuevo registro?

Indexed Sequential Access Method

Sparse Index-file:

¿Cómo insertar un nuevo registro?

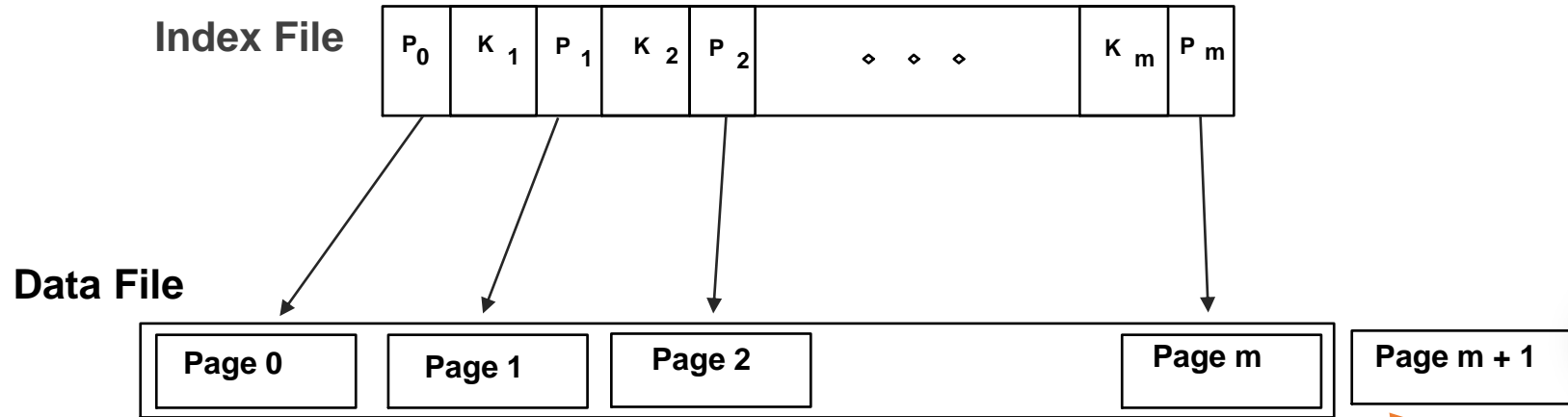


Se puede aplicar localización binaria en el archivo índice.

Indexed Sequential Access Method

Sparse Index-file:

¿Cómo insertar un nuevo registro?



Se puede aplicar localización binaria en el archivo índice.

Opción 2:

- Crear nueva pagina con un solo registro
- Encadenar nueva pagina a la pagina actual

Indexed Sequential Access Method

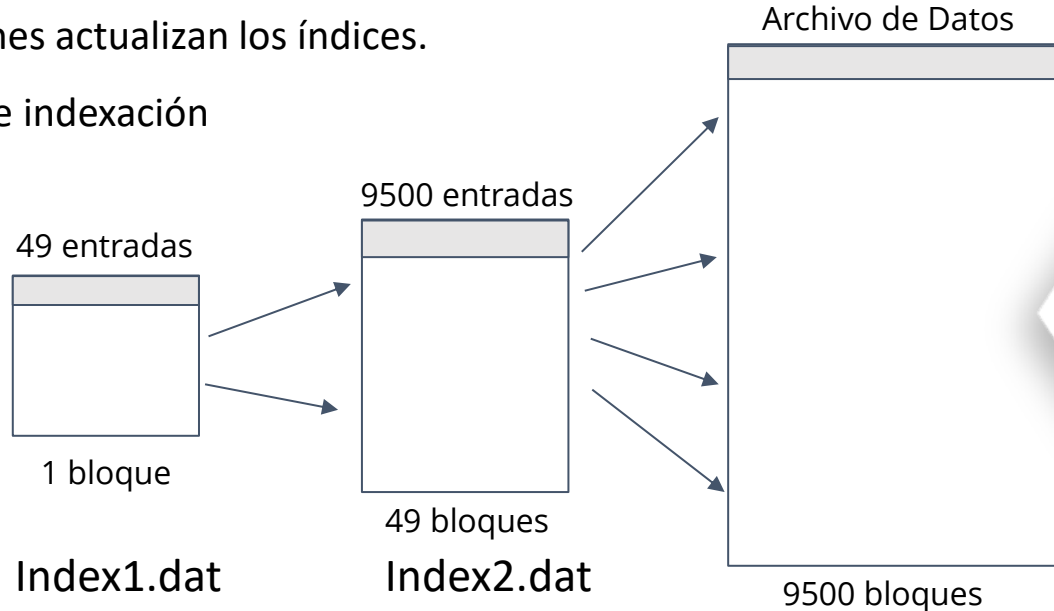
Sparse Index-File comparado a Dense Index-file

- Menos espacio y menos costo de mantenimiento para inserciones y eliminaciones.
- Generalmente más lento que el índice denso para localizar registros.
- **Good trade-off:**
 - El sparse index con una entrada de índice para cada bloque en el archivo de datos, correspondiente al menor valor de clave de búsqueda en el bloque (etiqueta del bloque).

Indexed Sequential Access Method

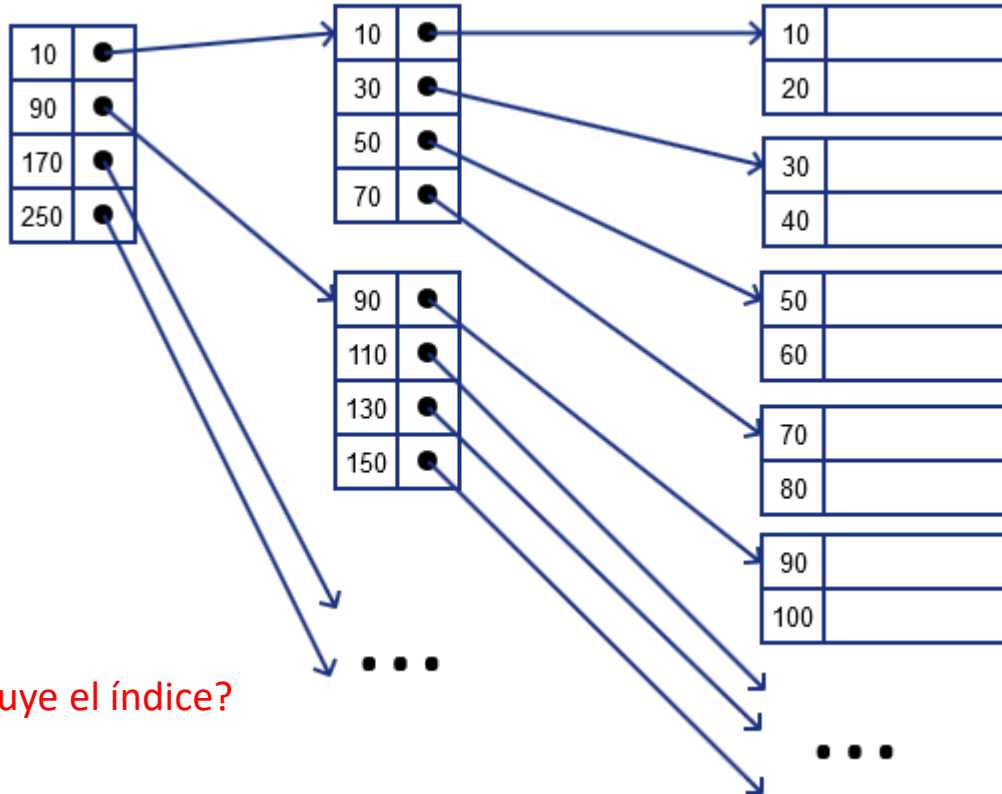
Multilevel Index-file:

- Si el index-file ocupa mucho espacio en memoria, entonces construir niveles sucesivos de índices hasta que el último quepa en un bloque.
- Inserciones y eliminaciones actualizan los índices.
- Búsquedas por niveles de indexación



Indexed Sequential Access Method

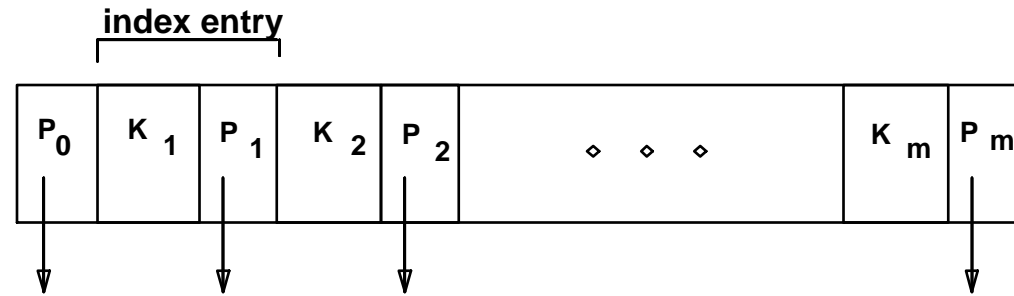
Multilevel Index-file:



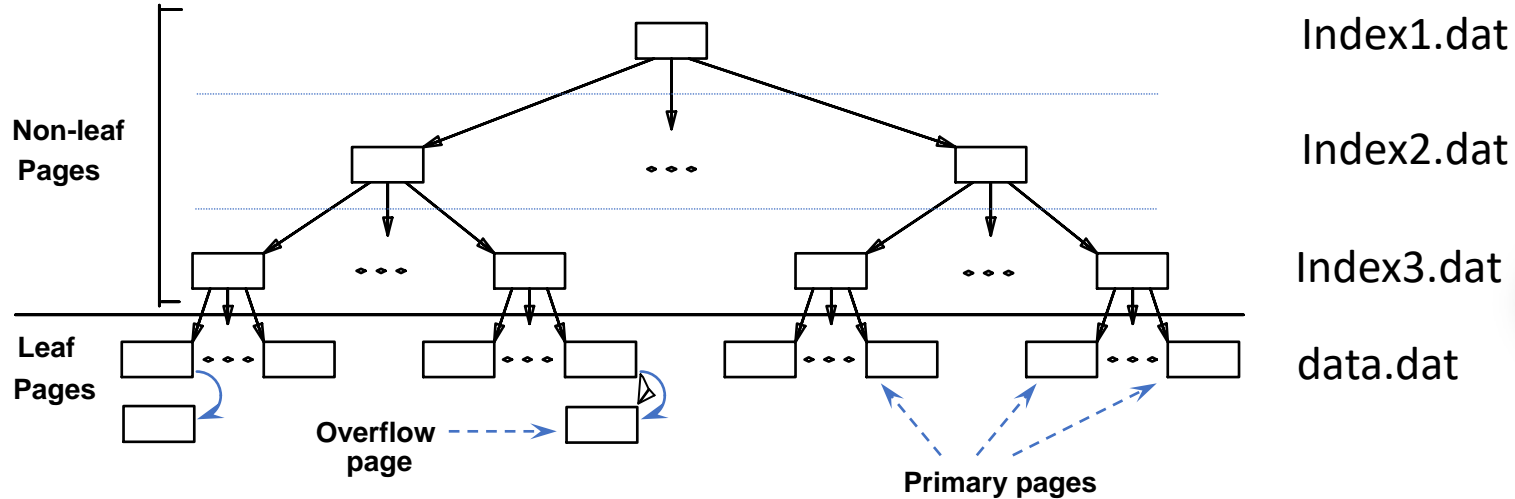
¿Como se construye el índice?

ISAM

Árbol Estático



- El archivo de índice puede ser bastante grande. ¡Pero podemos aplicar la idea repetidamente!



Leaf pages contain data entries.

ISAM

Árbol Estático

index1.dat

P_0	K_1	P_1	K_2	P_2	\diamond \diamond \diamond	K_m	P_m
-------	-------	-------	-------	-------	----------------------------------	-------	-------

index2.dat

P_0	K_1	P_1	K_2	P_2	\diamond \diamond \diamond	K_m	P_m
-------	-------	-------	-------	-------	----------------------------------	-------	-------

P_0	K_1	P_1	K_2	P_2	\diamond \diamond \diamond	K_m	P_m
-------	-------	-------	-------	-------	----------------------------------	-------	-------

- **Actividad Grupal.** Discuta y resuelva los siguientes algoritmos:
 - **Algoritmo para insertar un nuevo registro**
 - `void insert(T key, Record record);` //asumir que ya tenemos el índice construido
 - **Algoritmo para una búsqueda puntual**
 - `Record search(T key);`
 - **Algoritmo para una búsqueda por rango**
 - `vector<Record> search(T begin-key, T end-key);`
 - **Algoritmo para eliminar**
 - `Void remove(T key)`
 - *Indicar la complejidad computacional en función del número de acceso a memoria secundaria.*

Indexed Sequential Access Method

Multilevel Index-file (Static tree structure):

- **File creation:** Leaf (data) pages allocated sequentially, sorted by search key; then index pages allocated, then overflow pgs.
- **Search:** Start at root; use the key for comparisons to go to leaf.

$$\text{Cost} = \log_F N ; \quad F = \# \text{ entries/pg}; \quad N = \# \text{ leaf pgs}$$

- **Insert:** Find leaf that data entry belongs to, and put it there.
Overflow page if necessary.
- **Delete:** Find and remove from leaf; if empty page, de-allocate.

inserts/deletes affect only leaf pages.

Indexed Sequential Access Method

Ventajas:

- Búsquedas eficientes $O(L)$, L niveles de indexacion.
- Inserciones y eliminaciones eficientes,
 - Índice estático no requiere reconstruirse despues de cada inserción / eliminación.

Desventajas:

- Espacio extra para el mantener el index-file.
- El encadenamiento de excesivas paginas puede perjudicar el rendimiento del índice.

Laboratorio 2.2