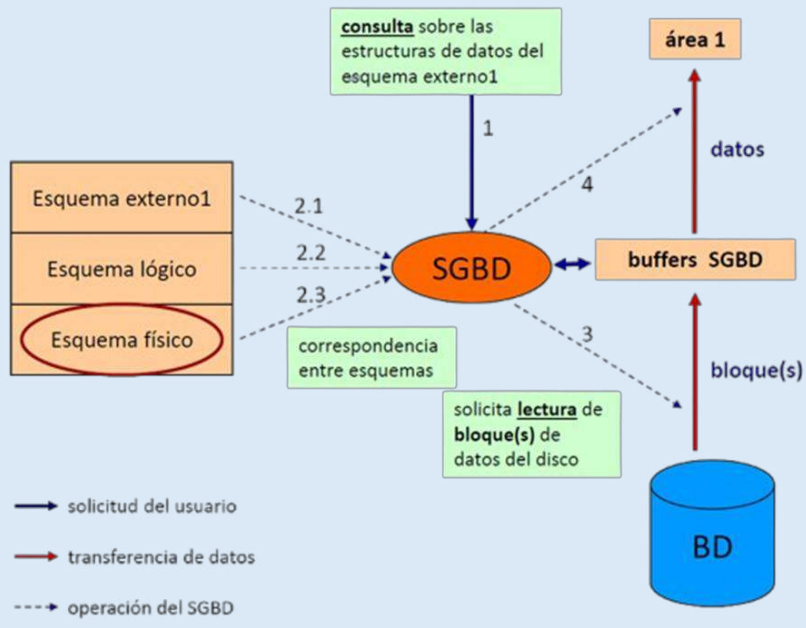
TEMA 5 – Implementación de Bases de Datos

1. Estructura física de las bases de datos

RECORDAMOS (TEMA 1)

Pasos que se siguen en un SGBD durante la ejecución de una operación de consulta.



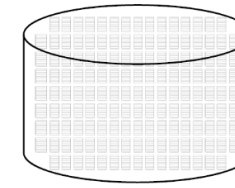
SGBD Objetivo 1: procesar correctamente transacciones en un entorno concurrente

- ✓ Las estructuras de datos de la BD se implementan como ficheros almacenados en MS:
  - **Fichero:** estructura de datos en MS que consiste en una secuencia de registros.
- ✓ El espacio en MS (disco) se divide en bloques:
  - **Bloque:** unidad de direccionamiento en disco y de transferencia de datos.
- ✓ Los bloques son transferidos a buffers de MP para consultar y actualizar los datos de la BD.
- ✓ Las tablas tienen **estructura de fichero** (filas → registros). Además existen una estructuras auxiliares, los **índices**.

SGBD Objetivo 2: procesar eficientemente transacciones en un entorno concurrente.

Factores que intervienen en el tiempo de respuesta en la ejecución de consultas/actualizaciones: gestión de los bloques en disco (factor variable), gestión de buffers (factor variables) en MP y gestión de los ficheros que implementan las estructuras de datos de la BD (factores fijos).

Gestión de bloques en disco

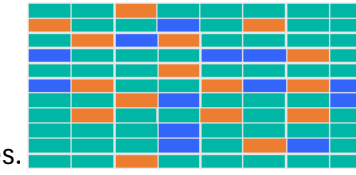


- **Bloque:** unidad de transferencia de datos
- **Operaciones:** adquirir y liberar bloques, leer y escribir bloques del disco.
- **Gestión de bloques libres en disco:** lista de bloques o mapa de bits.

Gestión de buffers en MP

- Partición de la MP en trozos (buffers) de un bloque.
- Transferencia de bloques entre disco y MP cuando sea necesario.
- **Gestión de buffers:**
  - Directorio de bloques almacenados en buffers.
  - Estrategia de actualización de la BD en disco (tema 3)
  - Gestión del bit de sucio y del bit reserva a de los bloques.

Buffers en MP



2. Organizaciones de ficheros

Registro: estructura de datos formada por la unión de varios elementos bajo una misma estructura. A cada elemento se le llama campo (o fila o tupla).

Fichero: secuencia de registros del mismo tipo (a excepción del *fichero mixto*).

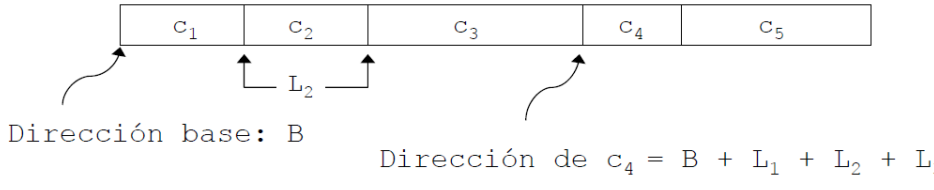
→ Ficheros con registros de longitud fija
→ Ficheros con registros de longitud variable

Fichero: características físicas en cabecera del fichero

1. **FORMATO DE REGISTRO:** orden de los campos, tipo de los campos (fijos/variables), longitudes de campos, caracteres separadores, valores nulos, etc.

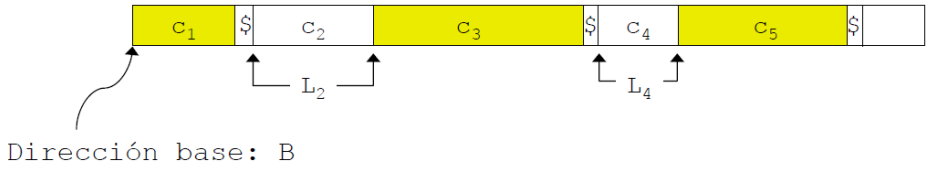
   Registros de longitud fija: los campos se almacenan en el orden en que aparecen en la definición del registro (esquema de la tabla).

C_i: campo iésimo del registro
L_i: longitud del campo c_i



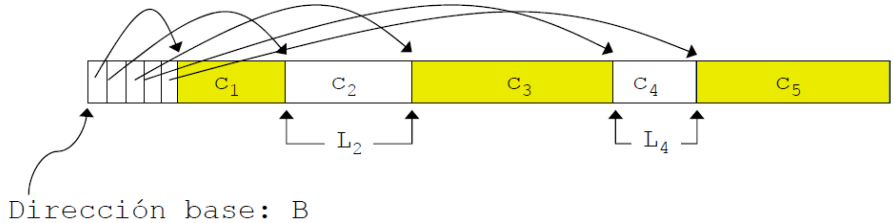
Registros de longitud variable: uso de un carácter separador $ que indica el fin de un campo.

C_i: campo iésimo del registro
L_i: longitud del campo de longitud fija c_i
$: carácter separador de los campos de longitud variable



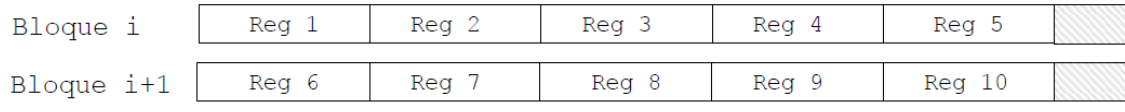
Registros de longitud variable: se guardan las direcciones (relativas) de los campos en el registro

C_i: campo iésimo del registro
L_i: longitud del campo de longitud fija c_i

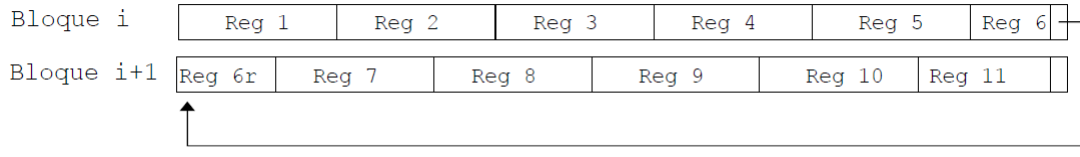


2. **FORMATO DE BLOQUE:** distribución y direccionamiento de registros en bloques del fichero.
   - Los registros se almacenan en los bloques del fichero.
   - Factor de bloque: $\text{fbl} = B \text{ DIV } L$ (número de registros que caben en un bloque)
     - Tamaño de registro de longitud fija: L
     - Tamaño del bloque: B
   - Espacio que no se puede usar en un bloque: $B - (\text{fbl} * L)$

Organización no extendida: un registro se almacena en un único bloque. Adecuada para registros de longitud fija, con $L < B$.



Organización extendida: un registro se puede almacenar en más de un bloque: un puntero en el primer bloque indica el bloque que contiene el resto del registro. Inevitable cuando $L > B$.



¿Cómo se localizan los registros en los bloques del fichero?

- ❖ Registros de longitud fija: compactación y mapa de bits.
- ❖ Registros de longitud variable: directorio de registros.
- ❖ Identificador del registro: $(Id\_bloque, nº\ registro)$

3. **GESTIÓN DE BLOQUES:** distribución de bloques en disco.
   - ➢ **Asignación continua:** los bloques del fichero son bloques contiguos en el disco.
   - ➢ **Asignación enlazada:** cada bloque del fichero contiene un puntero al siguiente bloque.
   - ➢ **Asignación de segmentos enlazados:** grupos de bloques contiguos enlazados por punteros.
   - ➢ **Asignación indexada:** un índice en la cabeza del fichero contiene las direcciones de los bloques del fichero.

Operaciones con ficheros: recuperación

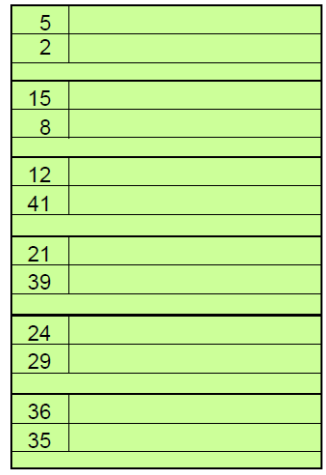
- ✓ **BuscarRegistro:** busca el primer registro que cumple una condición.
  - El bloque que contiene el registro es transferido a un buffer de memoria.
  - El puntero del registro del fichero apunta a ese registro en el buffer, pasando a ser el registro actual.
- ✓ **Leer:** copia el registro actual a una variable del programa del usuario.
- ✓ **BuscarSiguiente:** busca el siguiente registro (al registro actual) que cumple la condición de búsqueda.
  - El bloque que contiene el registro es transferido a un buffer de memoria.
  - El puntero del registro del fichero apunta a ese registro en el buffer, pasando a ser el registro actual.
- ✓ **BuscarTodos:** operación combinación de BuscarRegistro y BuscarSiguiente para todos los registros que cumplan la condición.
- ✓ **BuscarOrdenados:** buscarTodos en el orden de un campo del registro.

Operaciones con ficheros: actualización

- ✓ **Eliminar:** elimina el registro actual y (en algún momento posterior) transfiere el buffer que lo contiene a disco (puede significar marcarlo como borrado, no borrarlo físicamente).
- ✓ **Modificar:** modifica el valor de algunos campos del registro actual y (en algún momento posterior) transfiere el buffer que lo contiene a disco.
- ✓ **Insertar:** inserta un nuevo registro (en algún momento posterior) transfiere el buffer que lo contiene a disco.

2.1. Fichero desordenado

La secuencia de registros coincide con la secuencia de inserción (los nuevos registros se insertan al final del fichero. También se denomina **fichero de montículo o HEAP**.



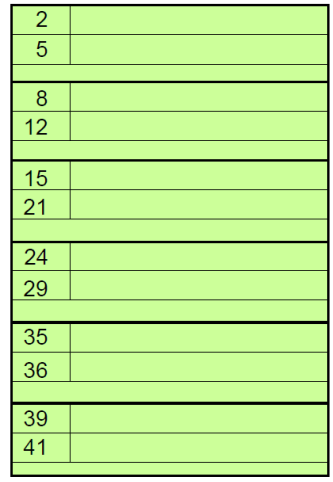
Operación	Coste
BuscarRegistro	Ineficiente (lineal)
BuscarOrdenados	Muy ineficiente
Insertar	Muy eficiente
Eliminar	Ineficiente
Modificar	Ineficiente

Registro de longitud fija: organización no extendida.
Formato de bloque: compactación de registros
fbl: 2 registros/bloque

2.2. Fichero ordenado

La secuencia de registros coincide con la ordenación de los registros por el valor de uno de sus campos, denominado **campo de ordenación**.

Si el campo de ordenación es clave (restricción de unicidad) se denomina **campo clave de ordenación**.



Operación	Coste
BuscarRegistro (campo de ordenación)	Eficiente (binaria)
BuscarRegistro (otro campo)	Ineficiente (lineal)
BuscarOrdenados (campo de ordenación)	Muy eficiente
Insertar	Muy ineficiente
Eliminar	Eficiente
Modificar	Eficiente

Registro de longitud fija: organización no extendida.
Formato de bloque: compactación de registros
fbl: 2 registros/bloque

2.3. Fichero disperso

Los registros se almacenan en bloques cuya dirección está determinada (cálculo) por el valor de un campo del registro: **campo de direccionamiento** (o dispersión).

Función de direccionamiento (o dispersión): función que se aplica al valor del campo de direccionamiento de un registro y devuelve la dirección del bloque donde se almacena el registro.

Las técnicas de direccionamiento calculado proporcionan una búsqueda muy eficiente por igualdad en el campo de direccionamiento.

Función de dispersión: f
Dominio del campo de dispersión K: D
Espacio de direcciones: $0 .. N-1$ (N bloques)

$$f : D \rightarrow 0 .. N-1$$

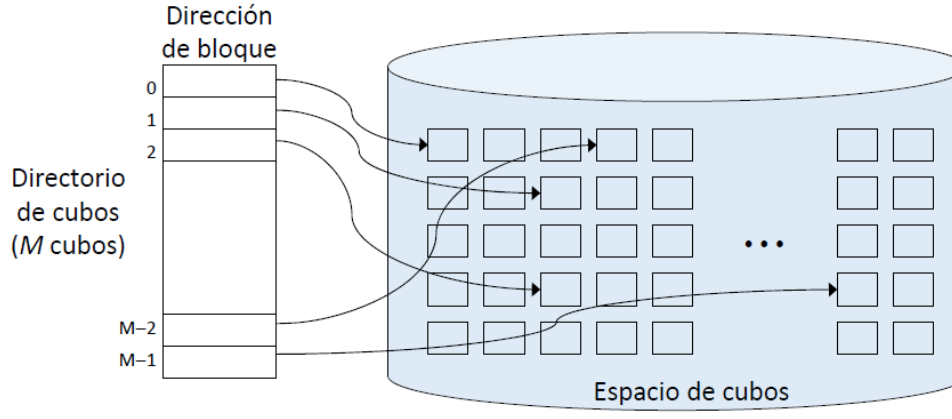
Direccionamiento calculado: espacio de direcciones

Espacio de direcciones: **cubos**

- El espacio en disco (bloques) se divide en cubos.
- Un cubo se compone de uno o varios bloques de discos contiguos.

La función de direccionamiento aplicada al campo de direccionamiento del registro devuelve el número de cubo del fichero donde se almacena.

Una tabla en la cabecera del fichero convierte el número de cubo en una dirección de bloque de disco (primer bloque del cubo).



Direccionamiento calculado: problemas

Colisión: la función de direccionamiento devuelve, para un nuevo registro, una dirección (cubo) que ya está llena.

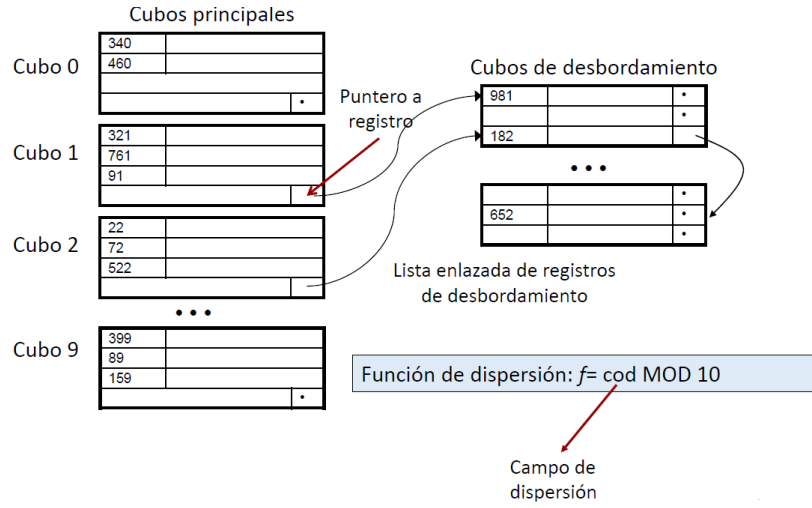
Espacio de direcciones fijo: se debe reservar a priori el espacio para el almacenamiento. Se debe estimar el espacio de direcciones (cubos) necesario:

- Si se estima por lo alto se desperdicia mucho espacio.
- Si se estima por lo bajo se producen muchas colisiones.

Direccionamiento calculado: colisiones

Solución:

- Se añade a los registros un nuevo campo: puntero a registro (dirección de bloque + posición relativa del registro en el bloque).
- Se mantiene en cada cubo un puntero a una lista enlazada de registros de desbordamiento del cubo.
- Los registros de desbordamiento se almacenan en cubos de desbordamiento.



Direccionamiento calculado: operaciones

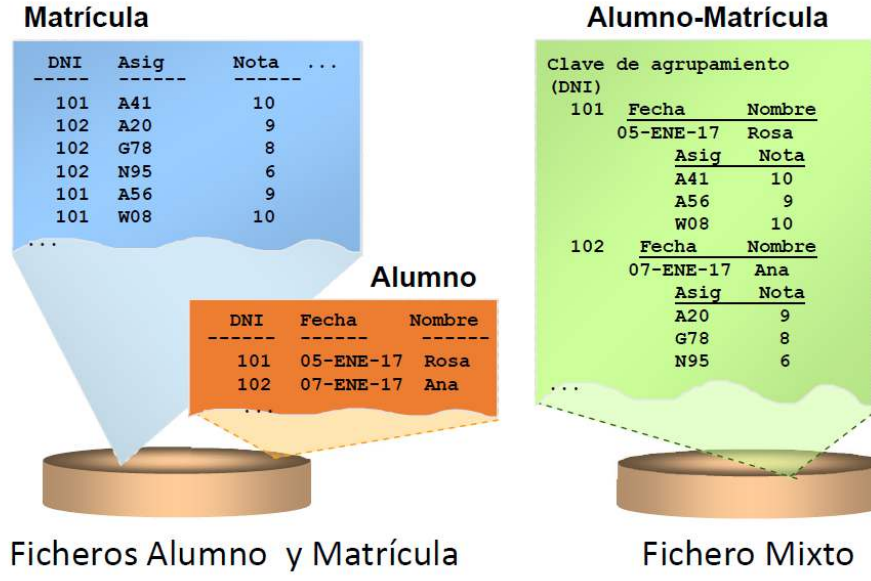
Operación	Coste
BuscarRegistro (por igualdad en el campo de direccionamiento)	Muy eficiente
BuscarRegistro (por otra condición distinta a la anterior)	Ineficiente (lineal)
BuscarOrdenados	Ineficiente
Insertar	Muy eficiente
Eliminar	Muy eficiente
Modificar	Muy eficiente

Si los cubos tienen lista de desbordamiento las operaciones pueden perder eficiencia.

2.4. Otras organizaciones: fichero mixto

Fichero en el que se almacenan registros de distinto tipo que están relacionados entre sí por el valor de un campo.

- ✓ Facilita el acceso conjunto a registros (de distinto tipo) relacionados entre sí.
- ✓ Para distinguir los registros de un fichero mixto, cada registro tiene, además de los valores de sus campos, un campo que indica el tipo de registro.
- ✓ Útil cuando se quieren almacenar registros de distinto tipo que están relacionados entre sí por el valor de algún campo y, además, es frecuente consultar conjuntamente los registros de ambos tipos relacionados.



En el ejemplo, se muestra un fichero con información sobre alumnos (*Alumnos)*, y otro fichero con información de en qué asignaturas se han matriculado y qué notas han obtenido.

Si cuando se consulta un alumno, se desea obtener los datos generales y también la información de sus notas, se tendrían que consultar los dos ficheros.

En este caso, el uso de un fichero mixto sería más adecuado. El fichero mixto *Alumno Matrícula* contendría registros de los dos tipos (*Alumno* y *Matrícula*), y en él se almacenarían consecutivamente los registros de ambos tipos relacionados por el campo DNI, con lo que la consulta sería más eficiente.

2.5. Modelo de costes

Modelo de costes: estimación del tiempo de ejecución (coste) de las operaciones en cada organización de fichero.

Las operaciones para las que se va a calcular el coste son las ya mencionadas anteriormente (**BuscarRegistro**, **BuscarTodos**, **Insertar**, **Eliminar**) más la operación **RecorridoCompleto**, que recupera todos los registros del fichero en una secuencia.

Fichero desordenado

- **RecorridoCompleto:** $B \cdot (D + F \cdot C)$
- **BuscarRegistro**
  - Mejor caso: $D + C$
  - Peor caso: $B \cdot (D + F \cdot C)$
- **BuscarTodos:** $B \cdot (D + F \cdot C)$
- **Insertar:** $2 \cdot D + C$
- **Eliminar:** $BuscarRegistro + D + C$

B: bloques de datos
F: factor de bloque
D: tiempo medio de L/E de un bloque.
C: tiempo medio de procesamiento de un registro en el bloque en MP
H: tiempo de cálculo de una función de hash

Fichero ordenado

- **Recorrido completo:** $B \cdot (D + F \cdot C)$
- **Buscar registro:**
  - Campo de ordenación:
    - Mejor caso: $D + C$
    - Peor caso: $\log_2 B \cdot (D + 2 \cdot C) + \log_2 F \cdot C$
- **BuscarSiguiente:**
  - Mejor caso: C
  - Peor caso: $D + C$
- **BuscarTodos:**
  - Campo de ordenación:
    - Mejor caso: $D + C + BuscarSiguiente \cdot nº\ coincidencias$
    - Peor caso: $\log_2 B \cdot (D + 2 \cdot C) + \log_2 F \cdot C + BuscarSiguiente \cdot nº\ coincidencias$
  - Otro campo: $B \cdot (D + F \cdot C)$

B: bloques de datos
F: factor de bloque
D: tiempo medio de L/E de un bloque.
C: tiempo medio de procesamiento de un registro en el bloque en MP
H: tiempo de cálculo de una función de hash

- **Insertar:** $BuscarRegistro + UbicarRegistro$
  - Mejor caso $UbicarRegistro$: $C + D$
  - Peor caso $UbicarRegistro$: $2 \cdot B \cdot (D + F \cdot C)$
- **Eliminar:** $BuscarRegistro + BorrarRegistro$
  - Mejor caso $BorrarRegistro$: $C + D$
  - Peor Caso $BorrarRegistro$: $2 \cdot B \cdot (D + F \cdot C)$

Fichero disperso

- **RecorridoCompleto:** $1{,}25 \cdot B \cdot (D + F \cdot C)$
- **Buscar registro por igualdad en el campo direccionamiento:**
  - Mejor caso: $H + D + C$
  - Peor caso: $H + D + F \cdot C$
- **BuscarRegistro por otra condición:** $BuscarRegistro$ en fichero desordenado
  - Mejor caso: $D + C$
  - Peor caso: $B \cdot (D + F \cdot C)$
- **BuscarTodos:** $1{,}25 \cdot B \cdot (D + F \cdot C)$
- **Insertar:** $BuscarRegistro + D + C$
- **Eliminar:** $BuscarRegistro + D + C$

B: bloques de datos
F: factor de bloque
D: tiempo medio de L/E de un bloque.
C: tiempo medio de procesamiento de un registro en el bloque en MP
H: tiempo de cálculo de una función de hash

Comparación mejor caso: accesos a bloque

Fichero \ Operación	Desordenado	Ordenado	Disperso
Recorrido Completo	B × D	B × D	1,25 × B × D
BuscarRegistro	D	D	D
BuscarTodos	B × D	D	1,25 × B × D
Insertar	2 × D	2 × D	2 × D
Eliminar	2 × D	2 × D	2 × D

Comparación peor caso: accesos a bloque

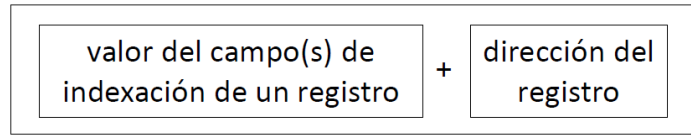
Fichero \ Operación	Desordenado	Ordenado	Disperso
Recorrido Completo	B × D	B × D	1,25 × B × D
BuscarRegistro	B × D	$\log_2$B × D	D
BuscarTodos	B × D	$\log_2$B×D + CosteSiguientes*	1,25 × B × D
Insertar	2 × D	2 × B × D	2 × D
Eliminar	B × D + D	2 × B × D	2 × D

* $CosteSiguientes$ = D × (nº de registros que cumplen la condición/factor de bloque)

3. Estructuras de índices

Índice: estructura de datos (en MS) que permite el acceso a los registros de un fichero por el valor de un campo(s): **campo(s) de indexación**.

Elementos de un índice (**entradas del índice**):



dirección del registro = dirección del bloque [+ dirección del registro (en el bloque)]

→ Las entradas del índice están ordenadas por el campo(s) de indexación.

→ Los índices permiten el acceso directo y el acceso ordenado a los registros del fichero por el campos(s) de indexación.

→ **Estructuras de datos utilizadas para construir índices:**

- Ficheros ordenados → **índices ordenados**
- Árboles de búsqueda → **índices en árbol B**

→ **Tipos de índice:**

- **Índice disperso:** el número de entradas del índice es menor que el número de registros
- **Índice denso:** el número de entradas del índice es igual al número de registros

3.1. Índices ordenados de un nivel

Índice ordenado: se construye como un fichero ordenado. Los registros de este fichero ordenado serán las entradas del índice, y la ordenación se definirá sobre el campo de indexación.

Caso A) Índice primario

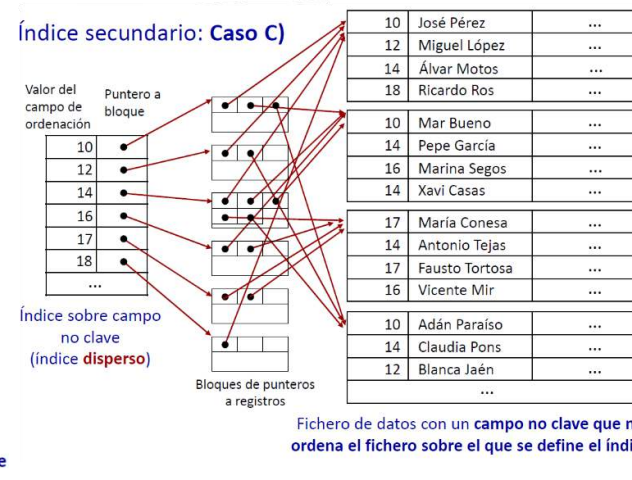
- **Fichero de datos:** fichero ordenado por un campo que es clave (restricción unicidad).
- **Índice:** definido sobre el campo de ordenación del fichero.

Caso B) Índice de agrupación

- **Fichero de datos:** fichero ordenado por un campo que no es clave.
- **Índice:** definido sobre el campo de ordenación del fichero.

Caso C) Índice secundario

- **Fichero de datos:** cualquier tipo de organización de fichero.
- **Índice:** definido sobre un campo del fichero que no es campo de ordenación.



3.2. Índices ordenados multinivel

Jerarquía de índices ordenados definidos sobre un fichero de datos:

- El primer nivel del índice puede ser de cualquier tipo (primario, de agrupación o secundario) y se define sobre el fichero de datos).
- En cada nivel (que no sea el primero) se define un índice primario (tipo A) sobre el índice del nivel anterior.
- El último nivel es aquel en el que todas las entradas del índice caben en un bloque.

fbl_{Idx}= 4 (fbl del índice)
fbl_F= 2 (fbl del fichero)
Bloques del fichero= 26/fbl_F= 13

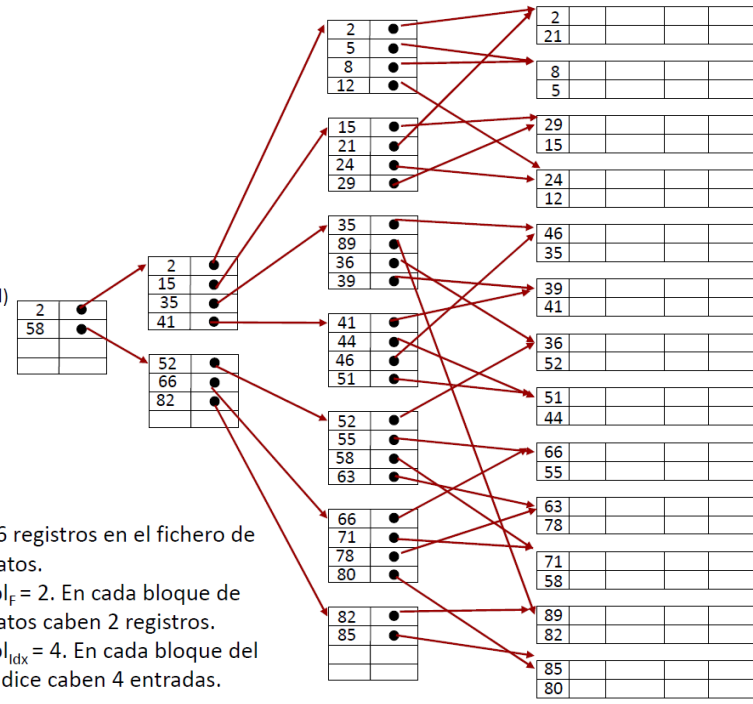
r_1 = 13 (entradas en 1er nivel)
$b_1 = \lceil r_1/fbl_{Idx} \rceil = \lceil 3{,}25 \rceil = 4$ (bloques en 1er nivel)

$r_2 = b_1 = 4$ (entradas en 2º nivel)
$b_2 = \lceil r_2/fbl_{Idx} \rceil = \lceil r_1/fbl_{Idx}^2 \rceil = 1$ (bloques en 2º nivel)

Altura del índice ***t*** = número de niveles:

$b_t = \lceil r_1/fbl_{Idx}^t \rceil = 1$

$t = \lceil \log_{fbl_{Idx}}(r_1) \rceil = \lceil 1{,}85 \rceil = 2$



fbl_{Idx}= 4 (fbl del índice)
fbl_F= 2 (fbl del fichero)

r_1 = 26 (entradas en 1er nivel)

Altura del índice ***t*** = número de niveles:

$b_t = \lceil r_1/fbl_{Idx}^t \rceil = \lceil 26/4^t \rceil$ 1

$t = \lceil \log_4(26) \rceil = \lceil 2{,}35 \rceil = 3$

- 26 registros en el fichero de datos.
- fbl_F = 2. En cada bloque de datos caben 2 registros.
- fbl_{Idx} = 4. En cada bloque del índice caben 4 entradas.

- ✓ El índice del primer nivel es de tipo A, B o C dependiendo de las características del fichero de datos.
- ✓ Cada nivel distinto del primero es un índice ordenado (tipo A) definido sobre el índice del nivel inmediatamente inferior.
- ✓ En cada nivel se divide por fbl_{Idx} el número de bloques, hasta llegar a un nivel en que todas las entradas del índice caben en un único bloque: estructura árbol (fbl_{Idx}: factor de bloque del índice).
- ✓ La búsqueda de un valor en el índice significa siempre el acceso a t (altura) bloques de índice (una rama del árbol).
- ✓ La búsqueda de un valor en un bloque del índice es binaria ya que son ficheros ordenados.
- ✓ La actualización del índice es muy costosa al ser ficheros ordenados.

3.3. Índices en árbol B

Se construyen como árboles de búsqueda. Pueden ser de dos tipos: **Árbol B** o **Árbol B⁺.**

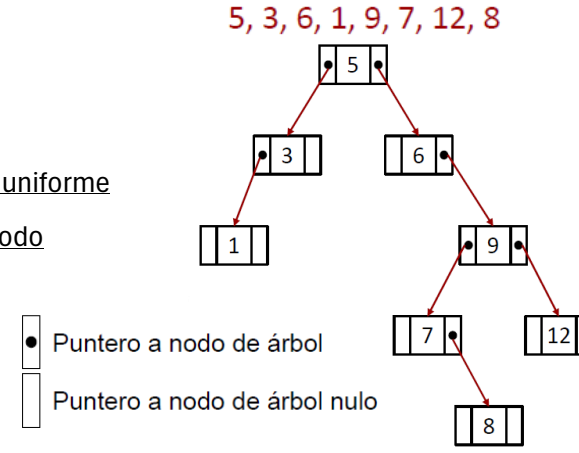
Los árboles B y B⁺ son **árboles n-arios de búsqueda equilibrados**, que garantizan una ocupación eficiente del espacio en los nodos.

- ➢ **Nodos:** raíz, nodo hoja, nodo interno, nodo padre y nodo hijo.
- ➢ **Orden del árbol:** número máximo de hijos de un nodo
- ➢ El **nivel de un nodo** es una unidad superior al de su padre (raíz: nivel 1).

- **Subárbol de un nodo:** un nodo hijo y todos sus descendientes.
- **Árbol de búsqueda:** árbol organizado de forma que facilita la búsqueda de un valor entre un conjunto de valores (almacenados en los nodos).

Árbol binario de búsqueda

- Árbol no equilibrado
- La búsqueda de un valor no es uniforme
- Ocupación: un valor en cada nodo

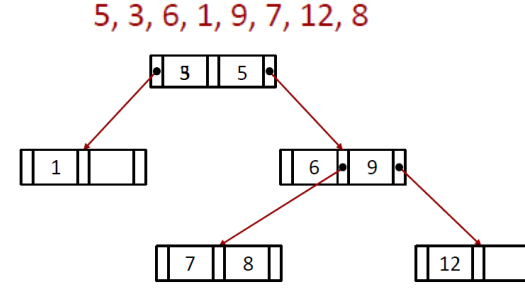


Árbol de orden p

- **Cada nodo tiene como máximo p hijos y $p-1$ valores**
- **Estructura:** $< P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q >$
  - $q \leq p$
  - P_i es un puntero a un nodo hijo o nulo
  - K_i es un valor de búsqueda único
- **Restricciones:**
  1. En cada nodo: $K_1 < K_2 < \cdots < K_{q-2} < K_{q-1}$
  2. Para todos los valores X en el subárbol al que apunta P_i, se cumple:
     - $K_{i-1} < X < K_i \ (1 < i < q)$
     - $X < K_1 \ , (i = 1)$
     - $K_{q-1} < X \ , (i = q)$

Árbol de búsqueda de orden 3:

- Árbol no equilibrado
- La búsqueda de un valor no es uniforme
- Ocupación de los nodos: no es uniforme



CONCLUSIÓN: se reduce el tamaño del árbol en comparación a los árboles binarios, pero siguen presentando problemas en la búsqueda de los valores y en la ocupación.

Estructura del índice ≅ Árbol de búsqueda de orden p

En los nodos se almacenan entradas de índice ($p - 1$ entradas) de la forma <valor, puntero>: valor del campo de indexación y puntero a registro o bloque del fichero de datos.

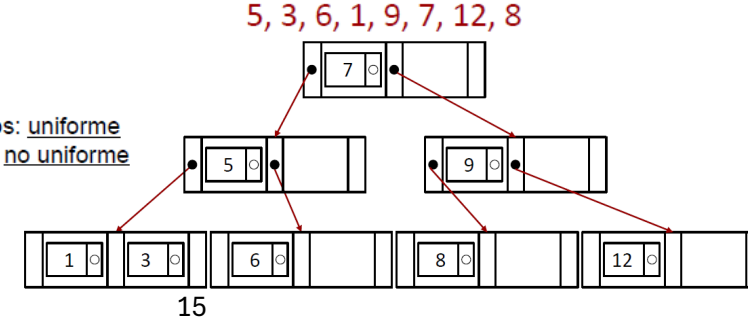
Almacenamiento en disco del índice:

- Cada nodo es un bloque de disco.
- El orden del índice dependerá del tamaño del bloque, del tamaño de los punteros y del tamaño del campo de indexación.

Árbol B

- Árbol equilibrado
- Ocupación eficiente: mínimo 50% para todos los nodos
- **Cada nodo tiene como máximo p hijos y $p - 1$ valores**
- **Estructura:** $< P_1, < K_1, Pr_1 >, P_2, < K_2, Pr_2 >, \dots, P_{q-1}, < K_{q-1}, Pr_{q-1} >, P_q >$
  - $q \leq p$
  - P_i es un puntero a un nodo hijo o un puntero nulo
  - $< K_i, Pr_i >$ es una entrada del índice:
    - K_i es un valor del campo de indexación
    - Pr_i es un puntero al registro cuyo valor del campo indexación = K_i
- **Restricciones:**
  1. En cada nodo: $K_1 < K_2 < \cdots < K_{q-2} < K_{q-1}$
  2. Para todos los valores X en el subárbol al que apunta P_i, se cumple:
     - $K_{i-1} < X < K_i \ (1 < i < q)$
     - $X < K_1, \ (i = 1)$
     - $K_{q-1} < X, \ (i = q)$
  3. Cada nodo, excepto el raíz y los hoja, tiene como mínimo $p/2$ punteros de árbol. El nodo raíz tiene como mínimo dos punteros a no ser que sea el único.
  4. Todos los nodos hoja están al mismo nivel y tienen la misma estructura que los internos, pero sus punteros de árbol son nulos.

- Árbol equilibrado
- Ocupación de los nodos: uniforme
- Búsqueda de un valor: no uniforme



CONCLUSIÓN: hemos solucionado el problema de la ocupación, pero la búsqueda de un valor sigue siendo no uniforme!!

Árbol B⁺

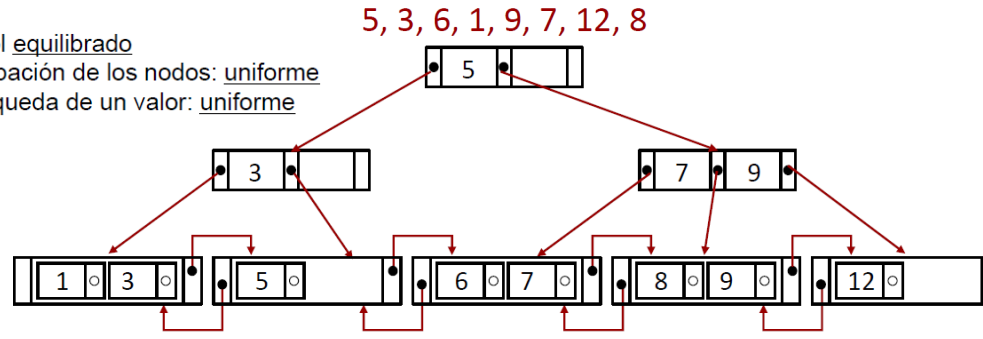
- Nodos con estructura distinta: nodos internos y nodos hoja
- Las entradas del índice sólo se almacenan en los nodos hoja
- Los valores en los nodos internos sirven para organizar la búsqueda
- Los nodos hoja están enlazados para favorecer la búsqueda

- **Cada nodo interno tiene como máximo p hijos y $p-1$ valores**
- **Estructura:** $< P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q >$
  - $q \leq p$
  - P_i es un puntero a un nodo hijo o un puntero nulo
  - K_i es un valor del campo de indexación
- **Restricciones:**
  1. En cada nodo interno: $K_1 < K_2 < \cdots < K_{q-2} < K_{q-1}$
  2. Para todos los valores X en el subárbol al que apunta P_i, se cumple:
     - $K_{i-1} < X \leq K_i \ (1 < i < q)$
     - $X \leq K_1, \ (i = 1)$
     - $K_{q-1} < X, \ (i = q)$
  3. Cada nodo interno tiene como mínimo $p/2$ punteros de árbol. El nodo raíz tiene como mínimo dos punteros de árbol si es un nodo interno.

- **Cada nodo hoja contiene entradas del índice**
- **Estructura:** $\ll K_1, Pr_1 >, < K_2, Pr_2 >, \dots, < K_{q-1}, Pr_{q-1} >, P_{siguiente}, P_{anterior} >$
  - $< K_i, Pr_i >$ es una entrada del índice:
    - Pr_i es un puntero a registro.
    - K_i es un valor del campo de indexación.
  - $P_{siguiente}$ es un puntero al siguiente nodo hoja del árbol.
  - $P_{anterior}$ es un puntero al anterior nodo hoja del árbol.

- **Restricciones:**
    4. En cada nodo hoja: $K_1 < K_2 < \cdots < K_{q-2} < K_{q-1}$
    5. Cada nodo tiene como mínimo $p/2$ entradas.
    6. Todos los nodos hoja están al mismo nivel.

- Árbol equilibrado
- Ocupación de los nodos: uniforme
- Búsqueda de un valor: uniforme



3.4. Análisis de los índices

✓ **Índice ordenado de un nivel**

- **Búsqueda:** búsqueda binaria en el índice
- **Búsqueda ordenada:** muy eficiente
- **Actualización del índice:** costosa

✓ **Índice ordenado multinivel**

- **Búsqueda:** muy eficiente (uniforme para todos los valores)
- **Búsqueda ordenada:** muy eficiente
- **Actualización del índice:** muy costosa

✓ **Índices en árbol B**

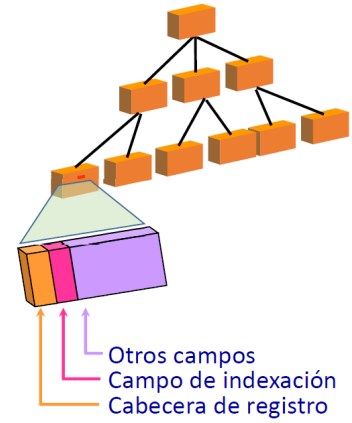
- Árboles de búsqueda: árboles B o B^+
- Los árboles B^+ pueden tener más entradas en sus nodos internos que los árboles B, por lo que pueden tener menos niveles para un mismo número de entradas.
    - **Búsqueda:** muy eficiente (uniforme en el índice B^+)
    - **Búsqueda ordenada:** muy eficiente (acceso a los nodos hoja en el B^+)
    - **Actualización del índice:** más eficiente que los índices ordenados

2.4. Fichero con estructura de índice

Los **índices** son estructuras de datos auxiliares (en MS) que facilitan el acceso directo y ordenado a los registros de un fichero por el valor de un campo(s) de indexación.

Si el tamaño de los registros de un fichero es pequeño y la mayoría de sus campos forman parte del campo de indexación, se puede implementar el fichero como un índice en árbol de búsqueda B, definido sobre el campo de indexación.

→ *En un fichero con estructura de índice se agiliza la búsqueda de registros por el valor del campo de indexación.*



4. Implementación de bases de datos relacionales

Diseño físico: se definen con precisión las estructuras de almacenamiento y los caminos de acceso en el SGBD que se van a utilizar.

Ajuste de la base de datos: análisis del rendimiento de la BD y modificación de las estructuras de almacenamiento, los caminos de acceso y la configuración del SGBD para lograr un funcionamiento eficiente.

Criterios para el diseño físico

- ✓ **Tiempo de respuesta (TR):** tiempo entre la puesta en marcha de una transacción y la respuesta. Factores que dependen del SGBD y otros que son generales al SO.
- ✓ **Aprovechamiento del espacio (AE):** espacio necesario para el almacenamiento de los ficheros y sus estructuras de acceso.
- ✓ **Productividad de las transacciones (PT):** número medio de transacciones que el sistema de BD puede procesar por unidad de tiempo.

Información para el diseño físico

- ✓ Datos promedio y "en el peor de los casos" de TR, AE y PT.
- ✓ Técnicas de obtención: analíticas, experimentales y prototipos o simulaciones.
- ✓ Ficheros: organización, tamaño del registro y número de registros.
- ✓ Patrones de actualización y consulta de las transacciones: datos consultados, volumen de datos, frecuencia de ejecución.
- ✓ Crecimiento de los ficheros: inserción de nuevos registros, alteración de la estructura del registro con inclusión de nuevos atributos.

4.1. Factores que influyen en el diseño físico

Análisis de consultas y transacciones

Para cada consulta:

1. Ficheros (tablas, relaciones, ...) a los que tendrá acceso
2. Atributos sobre las que se define la condición de selección
3. Atributos sobre las que se realizan operaciones de concatenación
4. Atributos que devolverá

→ *Los atributos de los puntos 2 y 3 son candidatos a la definición de estructuras acceso*

Para cada actualización:

1. Ficheros que se manipularán
2. Tipo de operación: inserción, modificación o borrado
3. Atributos sobre las que se define la condición de selección del borrado o modificación
4. Atributos que se alteran en una modificación

→ *Los atributos del punto 3 son candidatos a la definición de estructuras de acceso*

→ *Los atributos del punto 4 deben evitarse en la definición de estructuras de acceso*

Análisis de la frecuencia de invocación de las transacciones

- ✓ Para cada transacción (consulta/actualización): frecuencia de invocación esperada.
- ✓ Frecuencia esperada de uso de atributos: se consideran todas las transacciones, el tipo de atributo es selección o concatenación.
- ✓ Volumen de procesamiento alto: regla "80-20": apenas el 20% de las transacciones supone el 80% del procesamiento.

Análisis de las restricciones de tiempo sobre las transacciones

- ✓ Restricciones de rendimiento muy exigentes sobre algunas transacciones.
- ✓ Prioridades adicionales a los atributos que usan estas transacciones.
- ✓ Candidatos de mayor prioridad para las estructuras primarias de acceso.

Análisis de las restricciones de unicidad

- ✓ La comprobación de estas restricciones exige la consulta frecuente.
- ✓ Atributos sobre los que existen definidas restricciones de unicidad son candidatos a estructuras de acceso.

4.2. Decisiones de diseño físico

Alternativas de implementación

- ✓ **Ficheros de datos:** desordenado, ordenado o disperso.
- ✓ **Agrupaciones** de ficheros de datos.
- ✓ **Índices:** de tipo primario (A), de agrupación (B) y secundario (C).

Decisiones sobre ficheros de datos

- ✓ **Desordenado:** la opción más habitual, es adaptable a las circunstancias y se pueden definir los índices secundarios que sean necesarios.
- ✓ **Ordenado:** actualizaciones muy costosas, solo se utilizará si la información es muy estática.
- ✓ **Disperso:** es necesario tener una estimación del tamaño esperado del fichero y de las colisiones. Búsquedas más rápidas con igualdad, modificaciones de la clave de dispersión costosas. Se pueden definir los índices que sean necesarios.

Decisiones sobre ficheros mixtos

- ✓ Almacenamiento físico de registros de varios ficheros en el mismo bloque en función de algún atributo.
- ✓ Agiliza la manipulación combinada y entorpece la manipulación individual.
- ✓ Actualización del atributo de agrupación: implican cambio de ubicación física.

- ✓ Se utilizará si la manipulación combinada es la más frecuente, la generación de la información combinada es costosa o la actualización del atributo de agrupación no es frecuente.

Decisiones índices

- ✓ Determinar los atributos candidatos teniendo en cuenta que favorecen las operaciones de selección o concatenación y entorpecen las actualizaciones.
- ✓ **Cuando indexar atributos:**
  - Intervienen en condiciones de igualdad o de rango.
  - Participan en concatenaciones.
  - Forman parte de claves únicas.
- ✓ **Índice primario (tipo A) o de agrupación (tipo B):**
  - Si la clave de indexación tiene restricción de unicidad entonces definir un índice primario, si no, de agrupación.
  - Máximo uno por fichero de datos (ordenado), si hay varias posibilidades hay que determinar el más frecuentemente utilizado.
  - Beneficia mucho las consultas de rango.
  - Las actualizaciones son ineficientes.
- ✓ **Índice en árbol B o B+:**
  - Son muy flexibles.
  - Mejores que los índices como ficheros desordenados.
  - Adecuados para condiciones de igualdad y de rango.

Implementación estándar de una base de datos relacional

- ✓ Tabla ➡ Fichero desordenado
  - Excepciones:
    - Acceso muy rápido
    - Conocimiento volumen

    } ➡ Fichero disperso
    - Tabla muy estática ➡ Fichero ordenado o
    - Tabla asociativa ➡ Fichero Organizado como Índice
- ✓ Atributos (restricción de unicidad) ➡ Índices B^+
- ✓ Atributos utilizados en:
  - Condiciones de igualdad (=)
  - Condiciones de rango ($\geq$, $\leq$, $<$, $>$)
  - Concatenaciones (claves ajenas)

  } ➡ Índices B^+