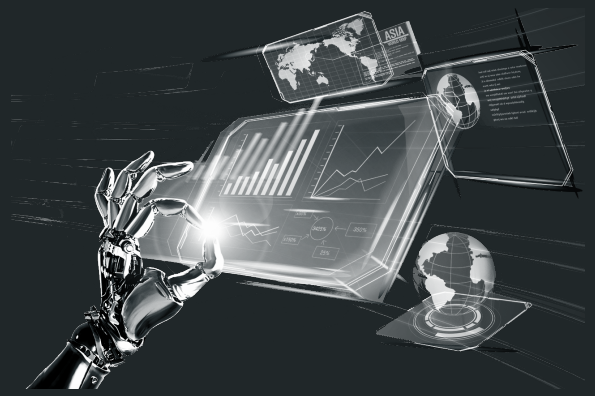


Gestión de Datos para Robótica

T3 - Estructuras de Almacenamiento e Indexación de Datos

Álvaro Vázquez Álvarez
Departamento de Electrónica e Computación

alvaro.vazquez@usc.es

Pabellón III - Despacho 4

Curso 2023-2024

Tabla de contenidos

Introducción

- La abstracción básica de los datos en un SGBD es un **archivo** o **colección de registros.**
- Cada **archivo** está compuesto de una o más **páginas**.
- Una **organización de archivo** es un método de organizar los registros en un archivo cuando se almacena en disco. Entender cómo están organizados los registros es esencial para utilizar un sistema de bases de datos de forma efectiva.
- Cada organización de archivo hace determinadas operaciones eficientes pero otras costosas. Por ejemplo, para recuperar los registros de un archivo de empleados en orden creciente de edad, la mejor organización es ordenar los registros de esa forma, pero:
  - El **mantenimiento es costoso** si el archivo se **modifica frecuentemente**.
  - Es necesario **recorrer todo el archivo** para encontrar los registros si se **combina con operaciones** sobre otros campos.
- La técnica de **indexación** puede ayudar cuando hay que acceder a una colección de registros de múltiples formas, además de permitir eficientemente varias clases de selección. Elegir una **buena colección de índices** es una de las herramientas más poderosas para **mejorar el rendimiento** de una base de datos.

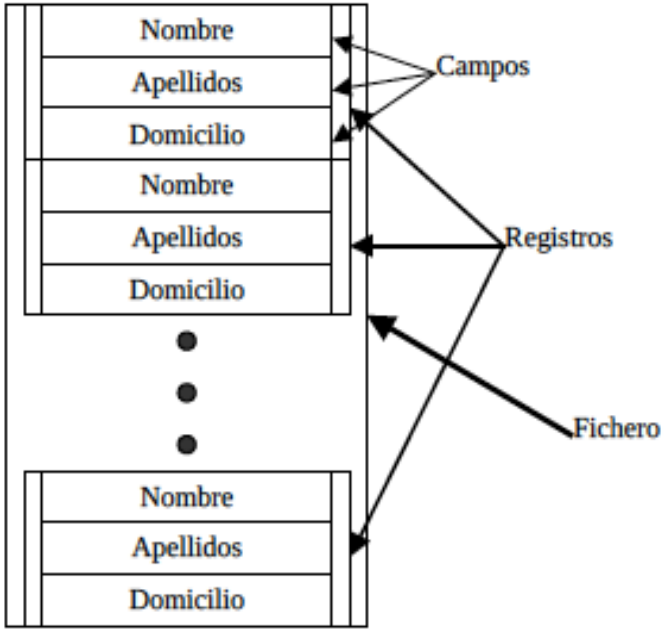
Estructuras de almacenamiento de datos

Objetivo: Determinar una organización de archivo eficiente para cada relación

- Los datos se almacenan en disco
- Página: unidad mínima de transferencia entre memoria y disco (usualmente 4KB o 8KB)
- Cada registro tiene un **identificador único de registro** (**idr**) con la dirección física de su página
- Coste de recuperación aproximadamente fijo por página. Sin embargo, si se leen varias páginas en el orden en el que están físicamente almacenadas, el coste puede ser mucho menos que el de leer las mismas páginas en un orden aleatorio.

Registros de datos

- Son las **unidades elementales** que componen un **archivo**.
- Un **registro** es una colección de información generalmente relativa a una entidad particular (un estudiante, un vehículo, etc.).
- Pueden contener varios datos, siendo cada uno de estos datos los **campos de los registros**.
- Un **campo** puede tener una **longitud fija o variable**, debiendo existir en el segundo caso alguna manera de indicar su longitud.
- Como consecuencia de ello, un **registro puede tener también una longitud variable**, refiriendo el término longitud del registro al tamaño máximo de un registro.



Organización de archivos de registros de datos

- HEAP (no ordenada / de montículo)
  - Disposición de registros ALEATORIA
  - Interesante cuando se recuperan todos los registros, o uno concreto por su **idr**
- ORDENADO (secuencial)
  - Registros ordenados por CLAVE DE BÚSQUEDA
  - Interesante cuando los registros se recuperan en cierto orden, o un rango de valores
- HASH
  - {buckets (cubos)} = {página 1ª [, página overflow]}
  - **H (clave búsqueda)** = nº bucket
  - Tipos: Estático/Dinámico

(Ramakrishnan, cap.9)

Índices. Definición

- Estructura auxiliar que ayuda a localizar datos de un archivo de registros bajo una condición

- Se asocia a una CLAVE DE BÚSQUEDA
  - $\forall$ campo puede $\in$ CLAVE BÚSQUEDA
  - CLAVE BÚSQUEDA $\neq$ Clave

- Está compuesto por ENTRADAS DE DATOS DEL INDICE
  - La entrada k^* localiza los registros de datos con CLAVE BUSQUEDA = k

Índices. Tipos

- ¿Qué se almacena en una ENTRADA DE DATOS DEL INDICE?

ALTERNATIVAS:

[1] ***k**** = registro de datos

[2] **<*k*, *rowid*** *de registro con clave de búsqueda* ***k*>**

[3] **<*k*, *lista-rowid*** *de registros con clave de búsqueda* ***k*>**

(las entradas de datos del índice son variables en longitud)

Índices. Tipos

ALTERNATIVAS:

[1] ***k**** = registro de datos

[2] **<*k*, *rowid*** *de registro con clave de búsqueda* ***k*>**

[3] **<*k*, *lista-rowid*** *de registros con clave de búsqueda* ***k*>**

(las entradas de datos del índice son variables en longitud)

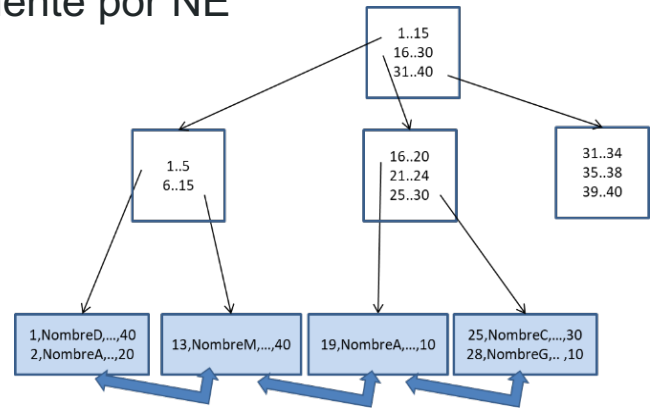
- [1] es una organización de archivos especial *(index-organized table)*
- [1] se implementa como un árbol B. Proporciona acceso rápido por la clave de búsqueda
- [1] se realiza un único acceso porque toda la información está en el índice
- [3] +compacta que [2], PERO... tamaño de registro variable

Ejemplo de índices

Ejemplo: Dado un archivo de datos con registros de la forma:

NE	NOMBRE	...	NDPTO
19	NombreA		10
2	NombreA		20
28	NombreG		10
25	NombreC		30
1	NombreD		40
13	NombreM		40

Alternativa [1] por el campo NE: Árbol B cuyas hojas contienen los registros de datos completos, ordenados secuencialmente por NE



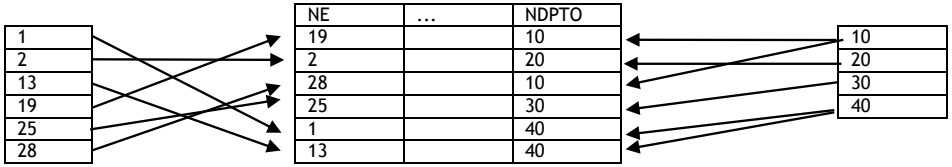
(Ramakrishnan, cap.9)

Ejemplo de índices

Ejemplo: Dado un archivo de datos con registros de la forma:

NE	NOMBRE	...	NDPTO
19	NombreA		10
2	NombreA		20
28	NombreG		10
25	NombreC		30
1	NombreD		40
13	NombreM		40

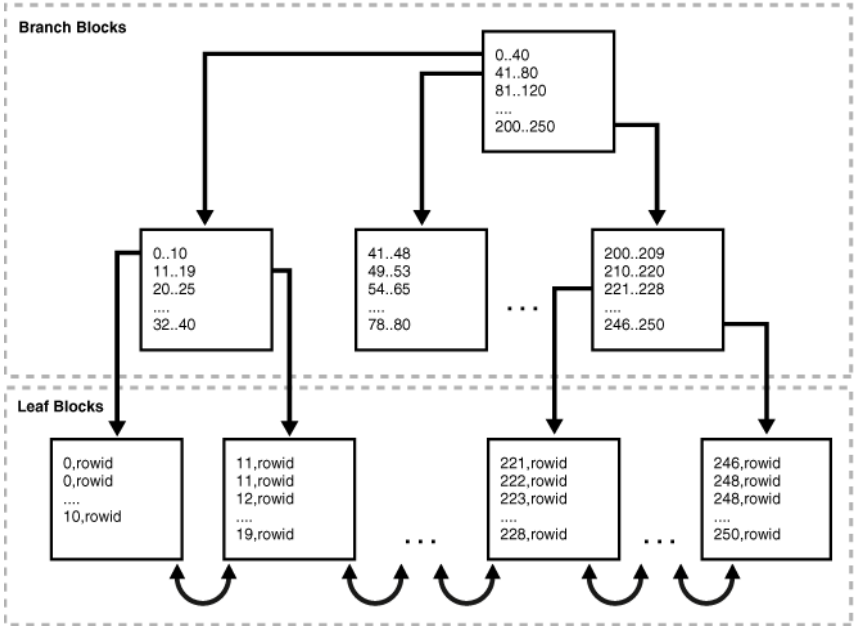
Alternativa [2] por el campo NE (izquierda) y Alternativa [3] por NDPTO (derecha)



Estructuras de datos de índices. Árbol B+

Árbol B+

- Balanceado, Ordenado
- Las entradas de datos del índice son las hojas
- Las entradas de datos del índice forman una lista doblemente enlazada
- Búsquedas por Rango, Igualdad



Estructuras de datos de índices. Árbol B+

- Nº operaciones de E/S = longitud (raíz a hoja) + nº páginas hoja con entradas que cumplen la condición

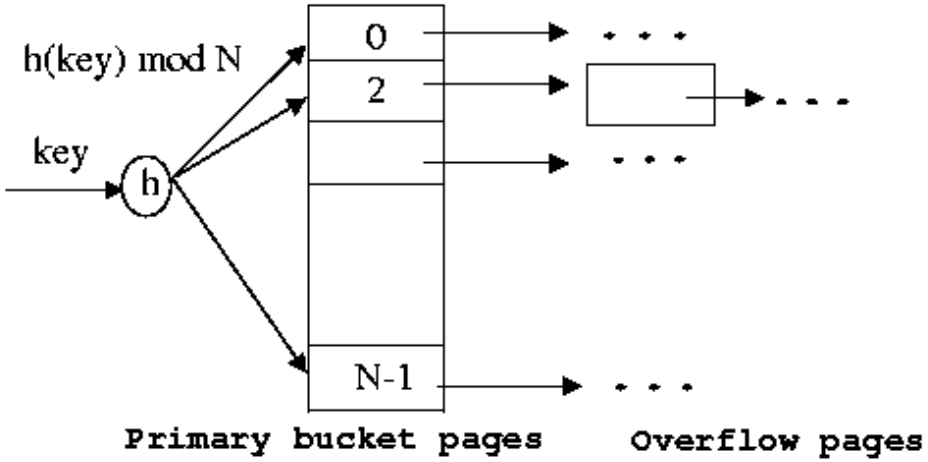
- La búsqueda en árbol B+ es más rápida que en archivo ordenado:
  - nº medio de punteros en cada nodo intermedio > 100:
  - altura del árbol suele ser 3 ó 4
  - ⇒ árbol de altura 4 contiene 100 millones de páginas hoja
  - ⇒ se necesitan 4 operaciones E/S para buscar en un archivo con 100 millones de páginas hoja

    (una búsqueda binaria llevaría $\log_2$ 100.000.000 > 25 operaciones E/S)

Estructuras de datos de índices. Función Hash

Basados en Hash

- {buckets} = {pág. 1ª [, pág. overflow]}
- H (clave búsqueda) = nº bucket
- NO Búsquedas por Rango



Estructuras de datos de índices. Función Hash

- Nº cubos (buckets) preasignado
  ⇒ para misma clave de búsqueda **k,** el hash calcula siempre la misma dirección de registro
- Número de entradas de índice aproximadamente el mismo en cada bucket
- Rendimiento se degrada debido al desbordamiento del bucket

Hashing estático

- Función Hash variable ⇒ se necesita un 2º paso para determinar la dirección de registro
- La cantidad de cubos cambia dinámicamente para adaptarse al crecimiento o reducción de los archivos de la BD
- Se reduce la degradación del rendimiento al minimizar el desbordamiento del bucket

Hashing dinámico

Estructuras de datos de índices. Índices bitmap

Índice bitmap

- Almacena un bitmap por cada valor de clave
- Cada entrada de datos del índice almacena punteros a varios registros de datos
- Cada bit en el bitmap se mapea a un rowid
  - Si el bit es 1, el registro contiene el valor clave

Ej:

```
SELECT ID, LAST_NAME, MARITAL_STATUS, GENDER
FROM   CUSTOMERS
ORDER BY id;
```

ID	LAST_NAME	MARITAL_STATUS	GENDER
1	Kessel		M
2	Koch		F
3	Emmerson		M
4	Hardy		M
5	Gowen		M
6	Charles	single	F
7	Ingram	single	F

Un índice bitmap para la columna Gender sería de la forma:

Value	Row1	Row2	Row3	Row4	Row5	Row6	Row7
M	1	0	1	1	1	0	0
F	0	1	0	0	0	1	1

Estructuras de datos de índices. Índices bitmap

Índice bitmap

- Se utiliza cuando:
  - Existe un nº elevado de registros de datos
  - El nº de valores distintos es menos del 1% del nº registros de datos (p.ej. género, estado civil,…)
  - El fichero es de solo lectura o estático
  - Las consultas son por igualdad con AND, OR y NOT

Ej: Supongamos una tabla con registros como se muestran a continuación:

CUSTOMER #	MARITAL_ STATUS	REGION	GENDER	INCOME_ LEVEL
101	single	east	male	bracket_1
102	married	central	female	bracket_4
103	married	west	female	bracket_2
104	divorced	west	male	bracket_4
105	single	central	female	bracket_2
106	married	central	female	bracket_3

Supongamos que existen índices bitmap para las columnas MARITAL_STATUS y REGION. Entonces, dada la consulta:

```
SELECT COUNT(*) FROM CUSTOMER
WHERE MARITAL_STATUS = 'married' AND REGION IN ('central', 'west');
```

Los índices bitmap pueden procesar esta consulta con gran eficiencia contando el número de unos en el bitmap resultante, como se muestra a continuación:

status = 'married'		region = 'central'		region = 'west'						
0		0		0		0		0		0
1		1		0		1		1		1
1	AND	0	OR	1	=	1	AND	1	=	1
0		0		1		0		1		0
0		1		0		0		1		0
1		1		0		1		1		1

Para localizar los clientes que cumplen la condición, bastará con utilizar el bitmap final para acceder a la tabla.

Clasificación de índices

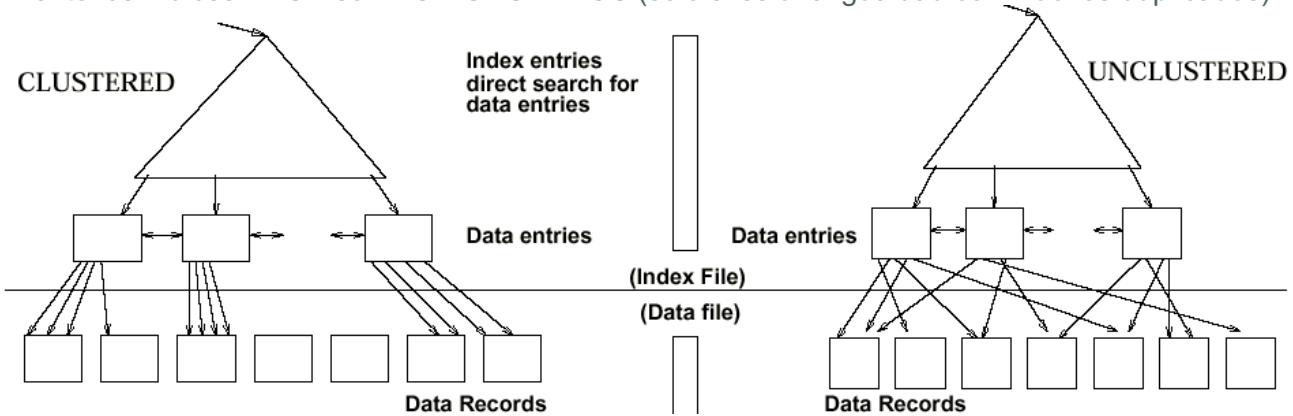
Únicos vs. No Únicos

- Único ⇒ garantiza que no contiene duplicados
- No único ⇒ puede contener duplicados

Clasificación de índices

Agrupados (CLUSTERED) vs. No Agrupados (UNCLUSTERED)

- Agrupado ⇒ orden reg. datos = orden entradas índice
- [1] ⇒ agrupado (PERO NO "⇐", aunque en la práctica suele serlo)
- [2][3] agrupados ⇒ reg. datos ordenados por la clave búsqueda
- máximo = 1 índice AGRUPADO por archivo de datos
- Coste de recuperación varía ENORMEMENTE si el índice está agrupado (en duplicados y rangos)
- Agrupado es muy útil en consultas POR RANGO o con muchos duplicados, PERO ES CARO
- Habitualmente los índices HASH son NO AGRUPADOS (solo si es una igualdad con muchos duplicados)

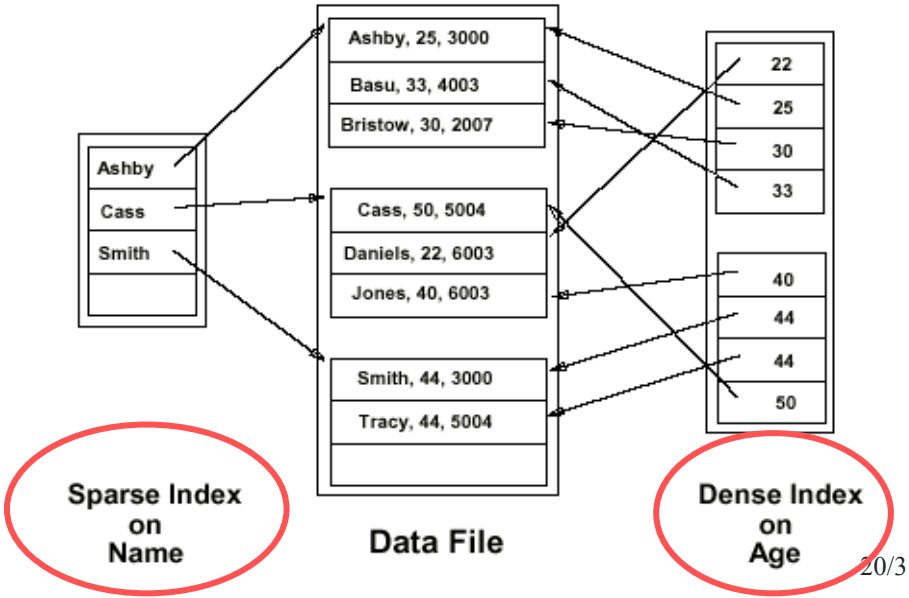


Clasificación de índices

Densos vs. Dispersos

- Denso ⇒ Existe una entrada de datos del índice por cada **registro** de datos
- Disperso (spare) ⇒ Existe una entrada de datos del índice por cada **página** de datos

  ⇒ AGRUPADO ⇒ máximo 1 índice DISPERSO por archivo de datos
- [1] ⇒ Denso
- [2] ⇒ Disperso / Denso
- [3] ⇒ Denso (generalmente)
- Los índices Hash siempre son densos



Clasificación de índices

Índice Compuesto

- Formado por varios campos
- Consulta por IGUALDAD ⇒ ej: *age = 20 and sal = 10*

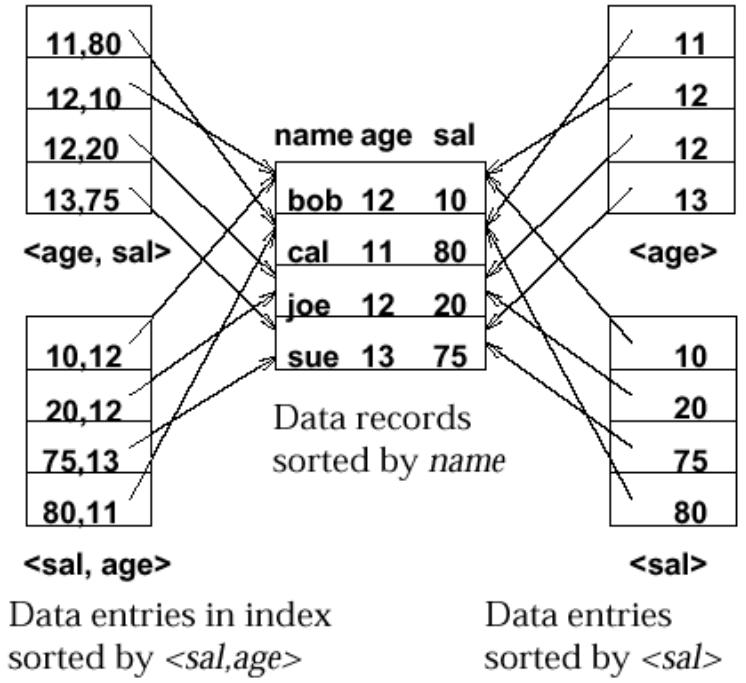
  ⇒ índice basado en HASH / B+
- Consulta por RANGO (+1) campo NO es constante ⇒ ej: *age < 30 and sal > 40*

  ⇒ ej: *age = 20*

  ⇒ índice B+
- El ORDEN de los CAMPOS en el índice es importante

  ⇒ ej: con la condición (*age = 30 and sal > 40)*

  <*age,sal*> da mejor resultado que <*sal,age*>



Pautas para la selección de índices

- Construir el índice SÓLO si hay consultas que se beneficien de él. Considerar el impacto en las actualizaciones !!!!!
- Los atributos del WHERE, GROUP BY, HAVING y ORDER BY son candidatos
  - Condición por igualdad sugiere índice HASH
    - Muy eficientes para joins y consultas exactas
  - Condición por rango exige índice en árbol B+
    - La agrupación es especialmente útil para consultas por rango o con muchos duplicados
- Elegir índices que beneficien a varias consultas
  - Como sólo se puede crear un índice agrupado por relación, elegirlo sobre las consultas más frecuentes y que involucren más tuplas
- Los índices compuestos son interesantes cuando WHERE / GROUP BY / ORDER BY contienen varias condiciones
  - Si hay selecciones por rango, ordenar los atributos del índice en base a la consulta

Ejemplos de índices simples

```
SELECT E.dno
FROM EMP E
WHERE E.hobby = 'Stamps'
```

EMP (eno, ename, dno, age, sal, hobby)
DEPT (dno, dname, mgr)

Campo	Estructura	Agrupado /No Agrupado	Denso / Disperso
E.hobby (si tuplas='Stamps' ↓↓)	Hash (igualdad)	NO Agrupado(↓↓duplicados, ordenación cara)	Denso (única opción)

Si se recuperan muchas tuplas (> 10%) la utilización de un índice No agrupado es muy ineficiente. Es mejor no hacer índice, recuperar todas las tuplas y filtrar

```
SELECT E.dno
FROM EMP E
WHERE E.eno = 552
```

Campo	Estructura	Agrupado /No Agrupado	Denso / Disperso
E.eno	Hash (igualdad)	NO Agrupado (se obtiene sólo 1 tupla)	Denso (única opción)

Ejemplos de índices simples

```
SELECT E.dno
FROM EMP E
WHERE E.age > 40
```

EMP (eno, ename, dno, age, sal, hobby)
DEPT (dno, dname, mgr)

Campo	Estructura	Agrupado /No Agrupado	Denso / Disperso
E.age (si tuplasage>40 ↓↓)	B+ (rango)	Agrupado (∃duplicados, consulta por rango)	Disperso (+ pequeño)
E.age (si tuplasage>40 ↑↑)	NO INDICE	No Agrupado sería INEFICIENTE. Agrupado sería MUY CARO	

El impacto de la agrupación depende del nº tuplas recuperadas:

- 1 tupla (i.e, igualdad sobre clave candidata) ⇒ No Agrupado
- Agrupación es interesante con ↑↑ duplicados
- ↑↑ tuplas (> 10%) ⇒ No Agrupado es + caro que recorrido secuencial

```
SELECT E.dno, COUNT(*)
FROM EMP E
WHERE E.age > 50
GROUP BY E.dno
```

Campo	Estructura	Agrupado /No Agrupado	Denso / Disperso
E.age (si tuplasage>50 ↓↓)	B+ (rango)	Agrupado (∃duplicados, consulta por rango)	Disperso (+ pequeño)
ó E.dno (si tuplasage>50 ↑↑)	Hash (igualdad) ó B+ (ordenado)	Agrupado (↑↑duplicados, los empleados se obtienen ordenadamente por departamento)	Denso (única opción para Hash) Disperso (+ pequeño)

Ejemplo utilizando múltiples índices

```
SELECT E.ename, D.dname
FROM EMP E, DEPT D
 WHERE E.sal BETWEEN 10000 AND 20000
        AND E.hobby='Stamps' AND E.dno=D.dno
```

EMP (eno, ename, dno, age, sal, hobby)
DEPT (dno, dname, mgr)

Campo	Estructura	Agrupado /No Agrupado	Denso / Disperso
D.dno	Hash (igualdad)	NO Agrupado (suponiendo pocas tuplas en la tabla)	Denso (única opción)
E.sal **y E.hobby**	B+ (rango) Hash (igualdad)	Agrupado (No clave, ∃ duplicados, consulta rango) NO Agrupado (sólo puede haber 1 índice agrupado por tabla)	Disperso (+ pequeño) Denso (única opción)

Si se crean *E.sal* **y** *E.hobby* el optimizador puede: 1) elegir el índice más selectivo, o 2) intersectar los identificadores de ambos índices

⇩

E.hobby debe ser NO Agrupado (no puede haber más de 1 índice agrupado por tabla y se prioriza el rango)

Ejemplo de pasos de ejecución

Describir los pasos que sigue el SGBD para resolver la siguiente consulta

```
SELECT E.dno, COUNT(*)
FROM EMP E
WHERE E.age > 50
GROUP BY E.dno
```

considerando los siguientes supuestos:

a) Existen pocos empleados con edad mayor de 50, por lo que se ha definido un índice <*E.age*> B+ agrupado disperso

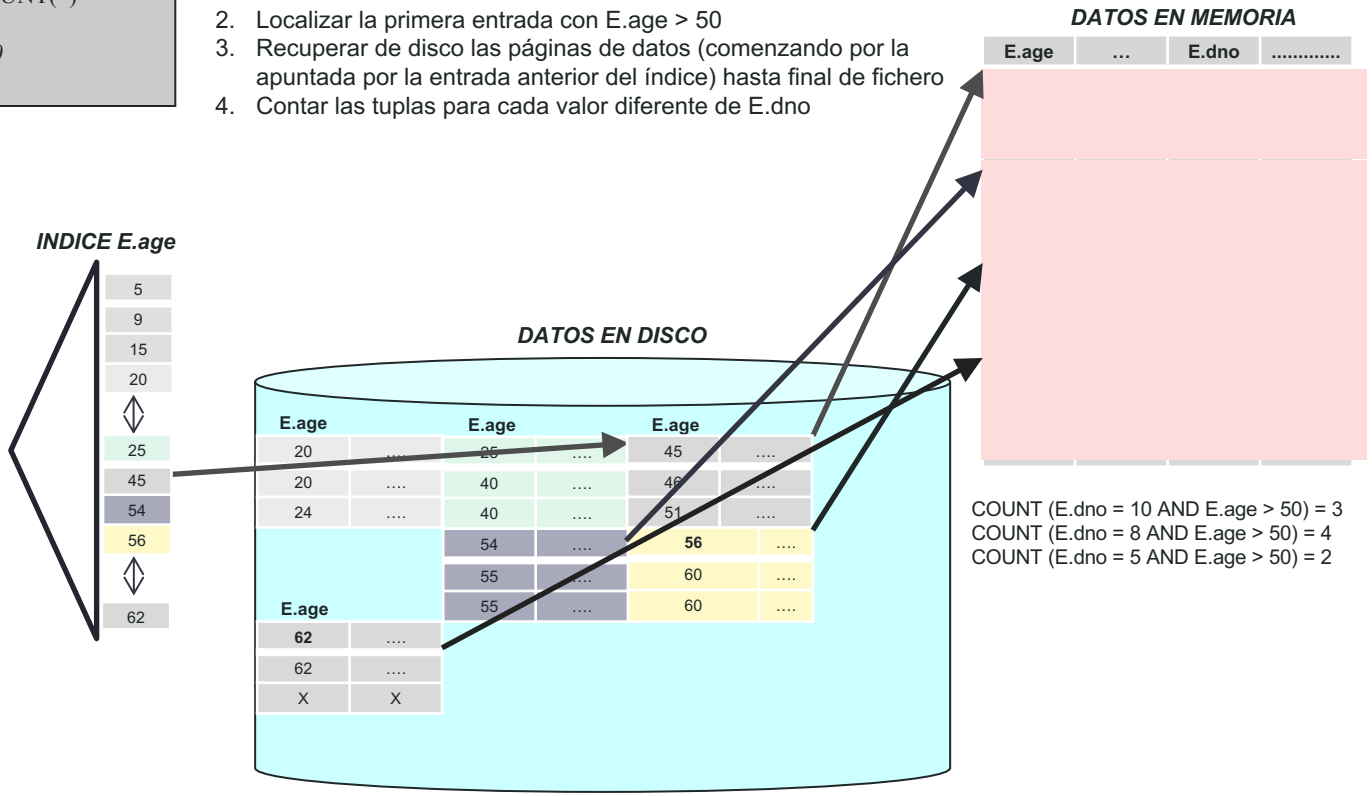
b) Existen muchos empleados con edad mayor de 50, y se ha definido un índice <*E.dno*> Hash agrupado denso

c) Existen muchos empleados con edad mayor de 50, y se ha definido un índice <*E.dno*> B+ agrupado disperso

A) Indice <E.age> B+ agrupado disperso

```
SELECT E.dno, COUNT(*)
FROM EMP E
WHERE E.age > 50
GROUP BY E.dno
```

1. Traer índice a memoria
2. Localizar la primera entrada con E.age > 50
3. Recuperar de disco las páginas de datos (comenzando por la apuntada por la entrada anterior del índice) hasta final de fichero
4. Contar las tuplas para cada valor diferente de E.dno

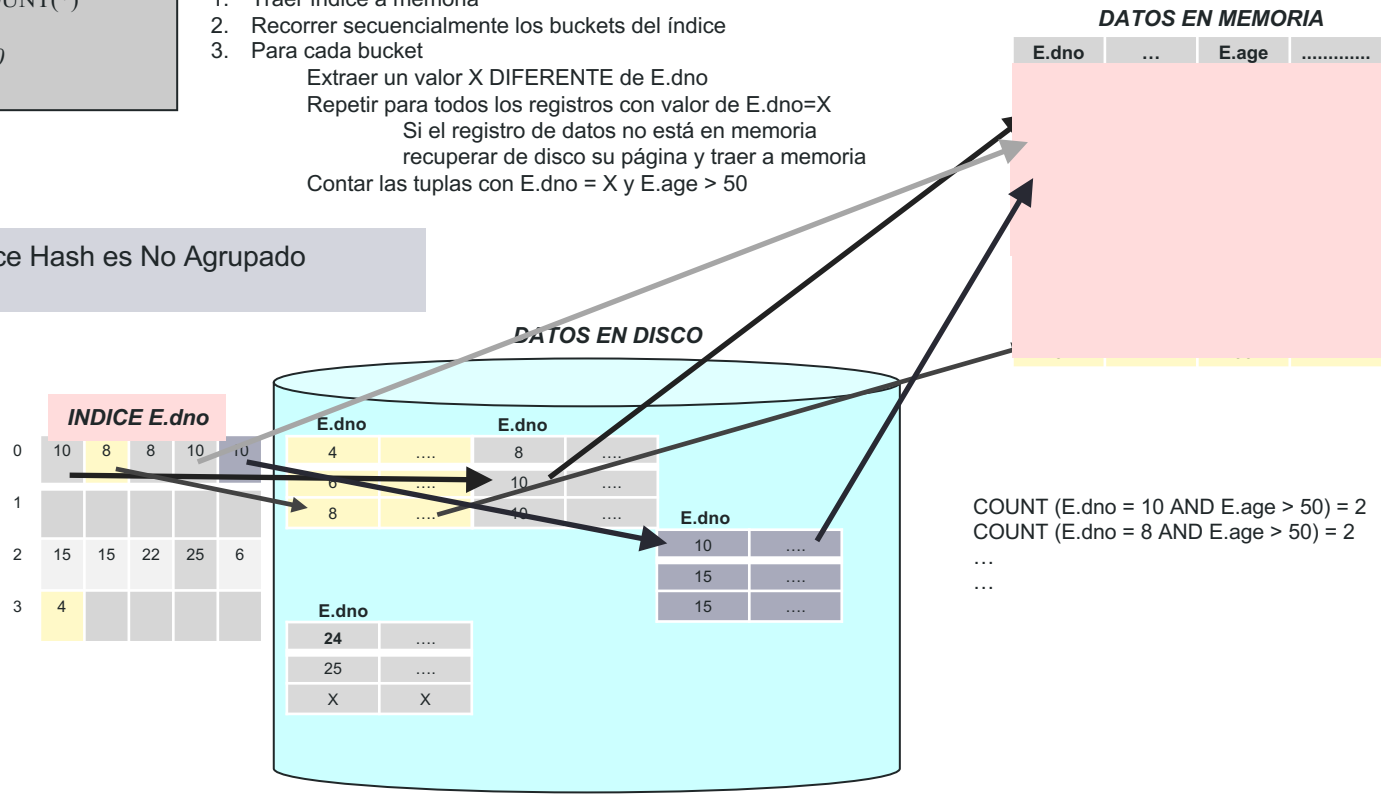


B) Indice <E.dno> Hash agrupado denso

```
SELECT E.dno, COUNT(*)
FROM EMP E
WHERE E.age > 50
GROUP BY E.dno
```

1. Traer índice a memoria
2. Recorrer secuencialmente los buckets del índice
3. Para cada bucket
   - Extraer un valor X DIFERENTE de E.dno
   - Repetir para todos los registros con valor de E.dno=X
     - Si el registro de datos no está en memoria
     - recuperar de disco su página y traer a memoria
   - Contar las tuplas con E.dno = X y E.age > 50

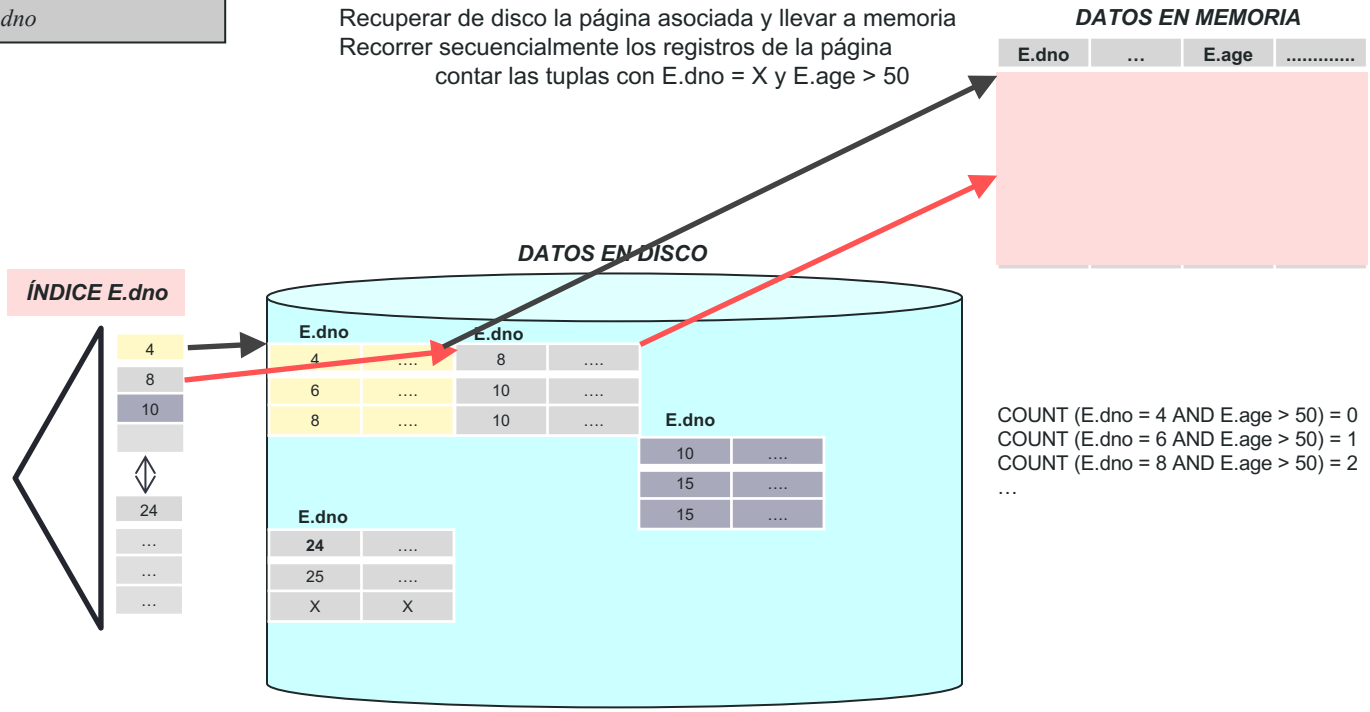
Habitualmente, un índice Hash es No Agrupado



C) Indice <E.dno> B+ agrupado disperso

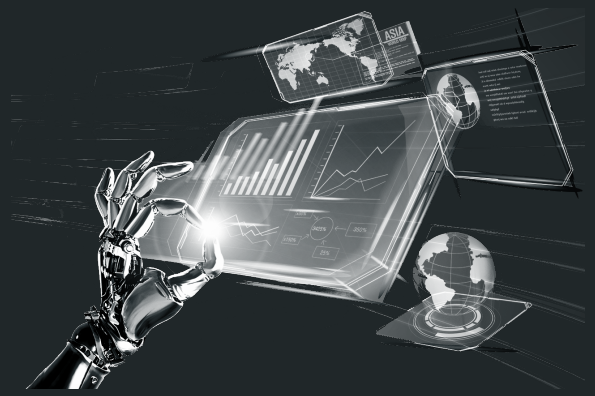
```
SELECT E.dno, COUNT(*)
FROM EMP E
WHERE E.age > 50
GROUP BY E.dno
```

1. Traer índice a memoria
2. Recorrer secuencialmente las entradas del índice
3. Para cada entrada del índice
   Recuperar de disco la página asociada y llevar a memoria
   Recorrer secuencialmente los registros de la página
   contar las tuplas con E.dno = X y E.age > 50



Bibliografía

- **Sistemas de Gestión de Bases de Datos (3ed). R. Ramakrishnan. McGraw-Hill (cap.9)**
- Fundamentos de Sistemas de Bases de Datos. Ramez A. Elmasri, Shamkant B. Navathe
- *Fundamentos de Bases de Datos.* A. Silberschatz. McGraw-Hill
- *Sistemas de Bases de Datos. Un Enfoque Práctico para Diseño, Implementación y Gestión*. T.Connolly, C. Begg, A. Strachan. Addison-Wesley



Gestión de Datos para Robótica

T3 - Estructuras de Almacenamiento e Indexación de Datos

Álvaro Vázquez Álvarez
Departamento de Electrónica e Computación

alvaro.vazquez@usc.es

Pabellón III - Despacho 4

Curso 2023-2024