

# Motivació Emmgatzematge 1

```
drop table bulto;
Create table bulto(a integer primary key, b char(60), c integer);

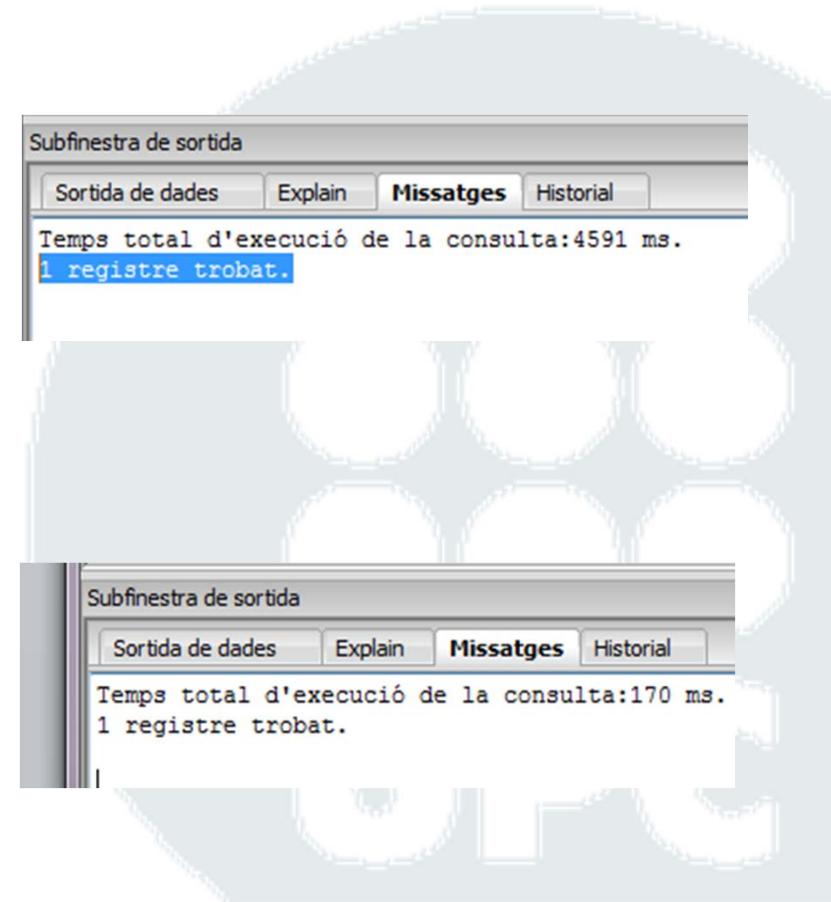
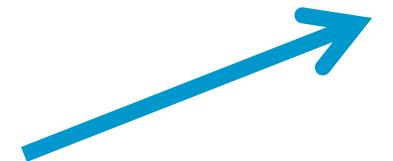
-- insertar 4000 tuples
CREATE OR REPLACE FUNCTION ins_4000() RETURNS void
AS $$
DECLARE
i INTEGER;
BEGIN
FOR i in 1 .. 4000 LOOP
insert into bulto values(i,i,i);
END LOOP;
END;
$$LANGUAGE plpgsql;

select * from ins_4000;
```

```
select count(*)
from bulto b1, bulto b2
where b1.c>10*b2.c
```

```
drop index prova;
create unique index prova on bulto(c) with (fillfactor= 80);
```

```
select count(*)
from bulto b1, bulto b2
where b1.c>10*b2.c
```



## Motivació Emmgatzematge 2

```

drop table bulto;
Create table bulto(a integer primary key, b char(60), c integer);

-- insertar 400000 tuples
CREATE OR REPLACE FUNCTION ins_400000() RETURNS void
AS $$
DECLARE
i INTEGER;
BEGIN
  FOR i in 1 .. 400000 LOOP
    insert into bulto values(i,i,i);
  END LOOP;
END;
$$LANGUAGE plpgsql;
  
```

`select * from ins_400000();`

`Explain`

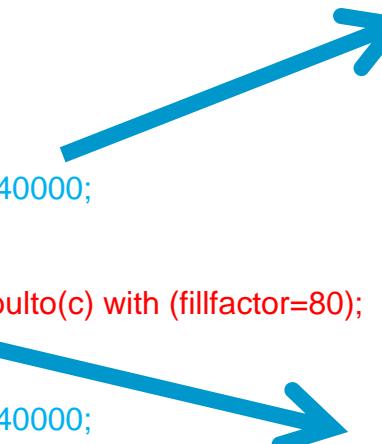
`Select * from bulto where c=240000;`

`drop index prova;`

`create unique index prova on bulto(c) with (fillfactor=80);`

`Explain`

`Select * from bulto where c=240000;`

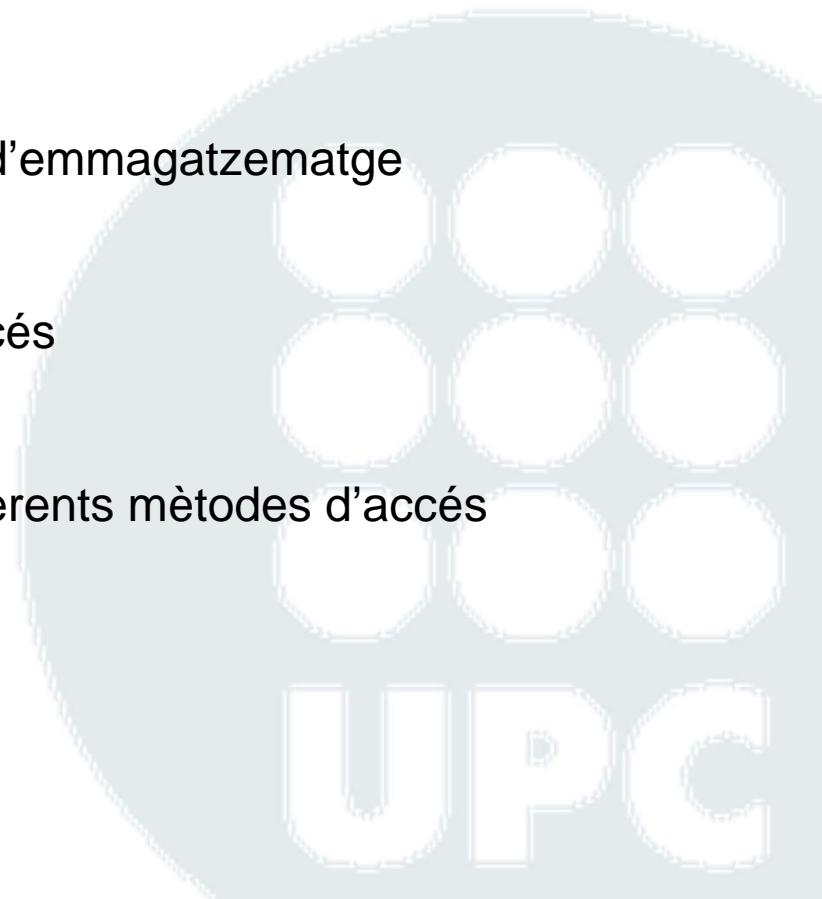


QUERY PLAN text	
1	Seq Scan on bulto (cost=0.00..9939.00 rows=1 width=69)
2	Filter: (c = 240000)

QUERY PLAN text	
1	Index Scan using idx on bulto (cost=0.00..8.76 rows=1 width=69)
2	Index Cond: (c = 240000)

## 9. Emmagatzemament i mètodes d'accés a les BD

- Memòria externa
- L'arquitectura dels components d'emmagatzematge
- Implementació de mètodes d'accés
- Annex: Càlcul de costos pels diferents mètodes d'accés



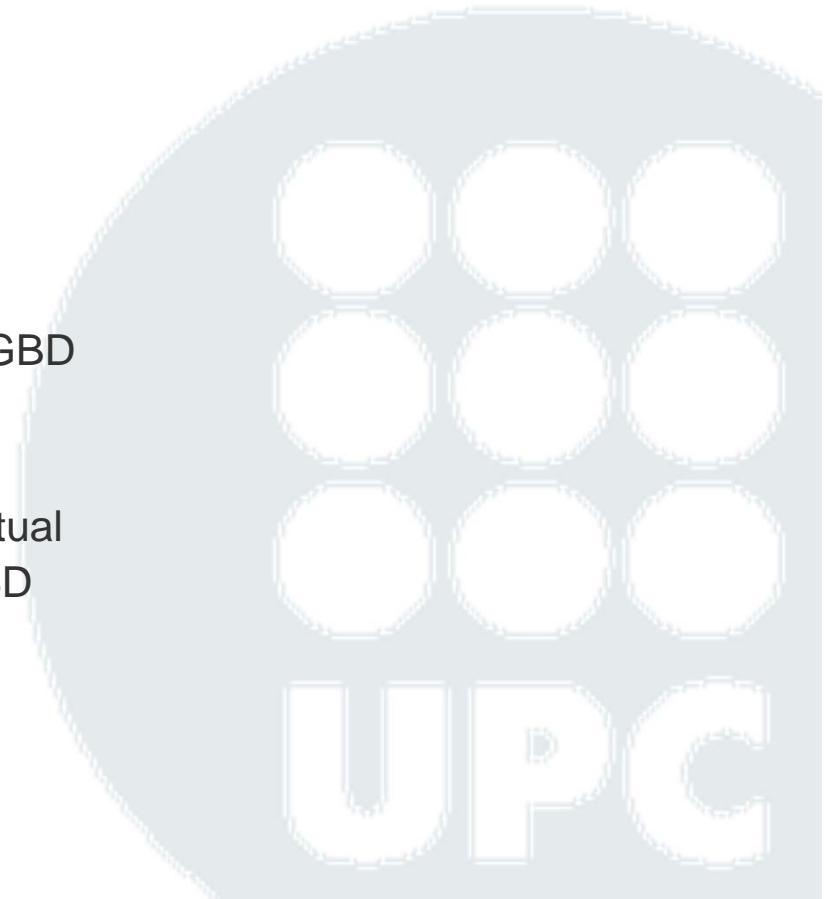
## Memòria externa: Necessitats

	mem. interna	mem. externa
preu	alt	baix
temps d'obtenció	constant ( $10^{-6}$ s a $10^{-9}$ s) micros a nanos	variable ( $10^{-3}$ s) milis
capacitat	poca	il.limitada
volatilitat	si	no volàtil

- La memòria externa és necessària perquè és **permanent** (no volàtil) i proporciona una gran **capacitat**.
- La memòria externa té el problema del **temps** d'obtenció de les dades.

## L'arquitectura dels components d'emmagatzematge

- Introducció i objectius
- Arquitectura dels components d'emmagatzematge
- El nivell físic
  - La pàgina
  - La fila
  - El camp
  - Gestió de la pàgina
  - Altres tipus de pàgines
  - Extensió i fitxer
  - Entrades/sortides en SGBD
- El nivell virtual
  - Justificació, definició
  - Estructura de l'espai virtual
  - Adreçament en un SGBD
  - Record IDentifier (RID)
  - Tipus d'espais virtuals
- Altres aspectes



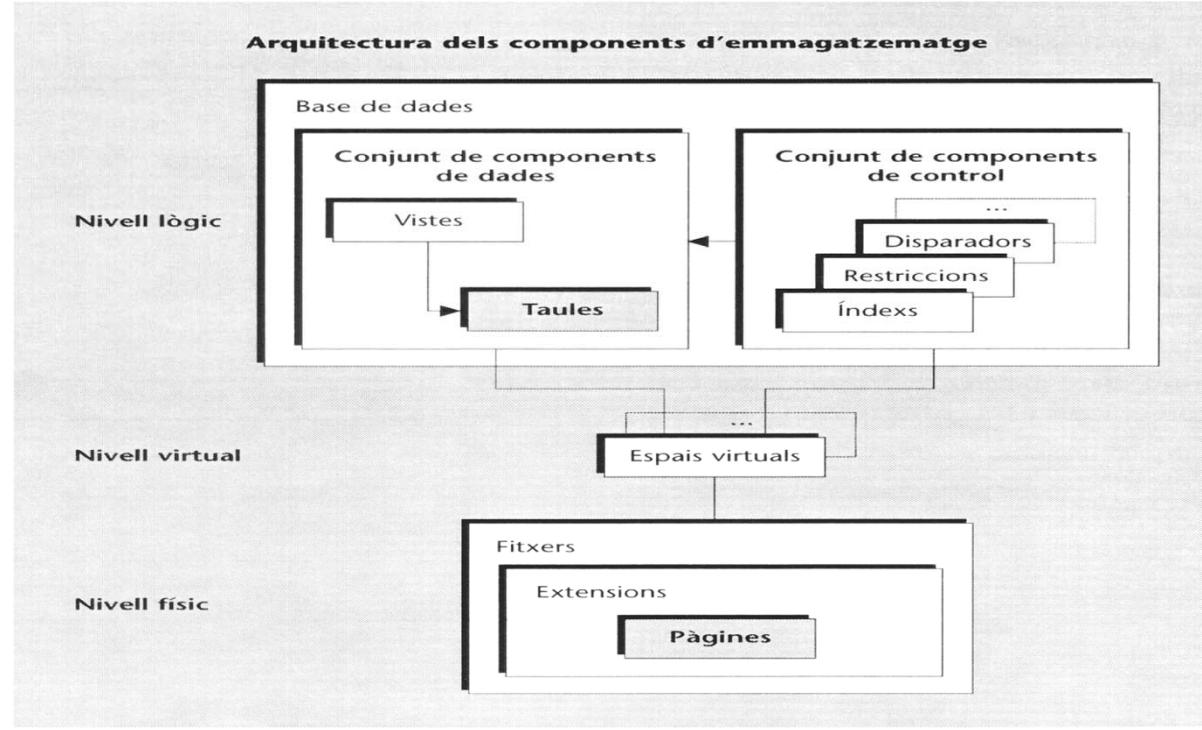
## Arquitectura dels components d'emmagatzematge: Introducció i objectius

- Els components lògics s'han d'emmagatzemar en un suport no volàtil -> cal garantir-ne l'accessibilitat.
- Cada sistema segueix un model diferent (no hi ha estàndard per les qüestions físiques) però existeixen uns patrons similars si deixem de banda els detalls.
  - “màxim comú divisor” dels components usats en els sistemes comercials
- **Objectius General**
  - Tot i que les dades d'una base de dades és guarden en fitxers, aquests segueixen uns patrons més complexos que els dels fitxers simples. L'objectiu és conèixer aquests patrons, per tal de poder fer un disseny físic d'una base dades més acurat.
- **Objectius Concrets**
  - Distingir els components i poder situar-los en una arquitectura funcional
  - Entendre el concepte de pàgina, la seva estructura, agrupacions
  - Entendre la funcionalitat, l'estructura i els tipus d'espai virtual

## Arquitectura dels components d'emmagatzematge

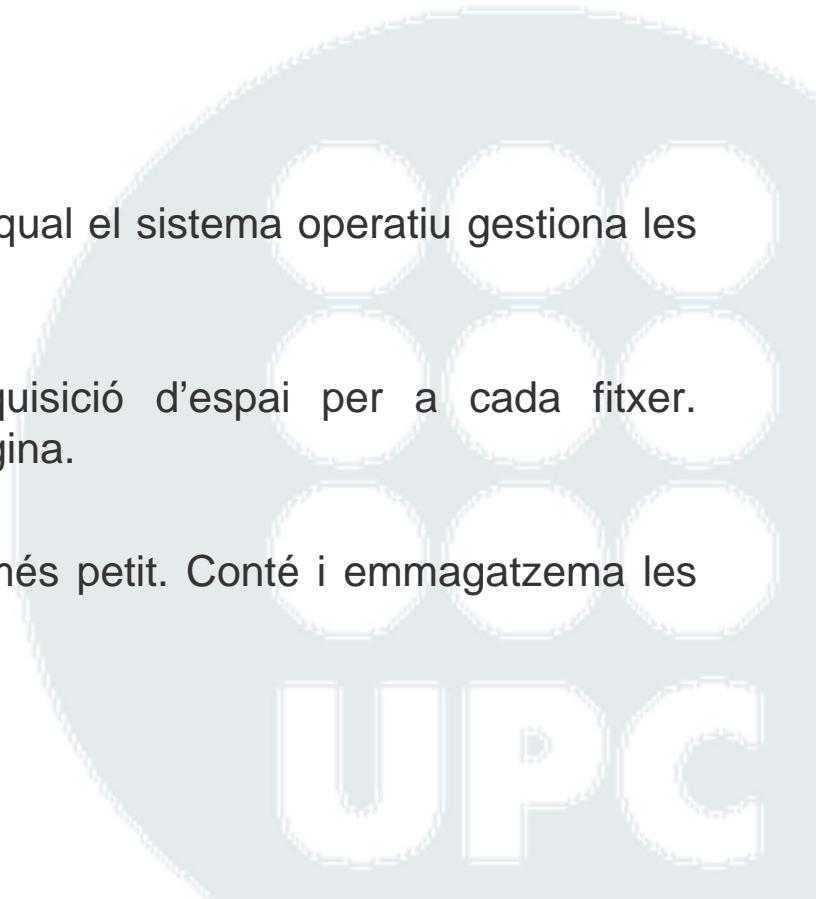
- BD: Conjunt de dades persistents!
  - No han de desaparèixer entre execucions
  - Emmagatzemades en un medi no volàtil (disc magnètic)
- L'arquitectura de tres nivells (lògic, virtual, físic) ens serveix per comprendre la relació entre les dades tal i com les veu el programador o usuari final i tal com estan emmagatzemades:

Figura 1



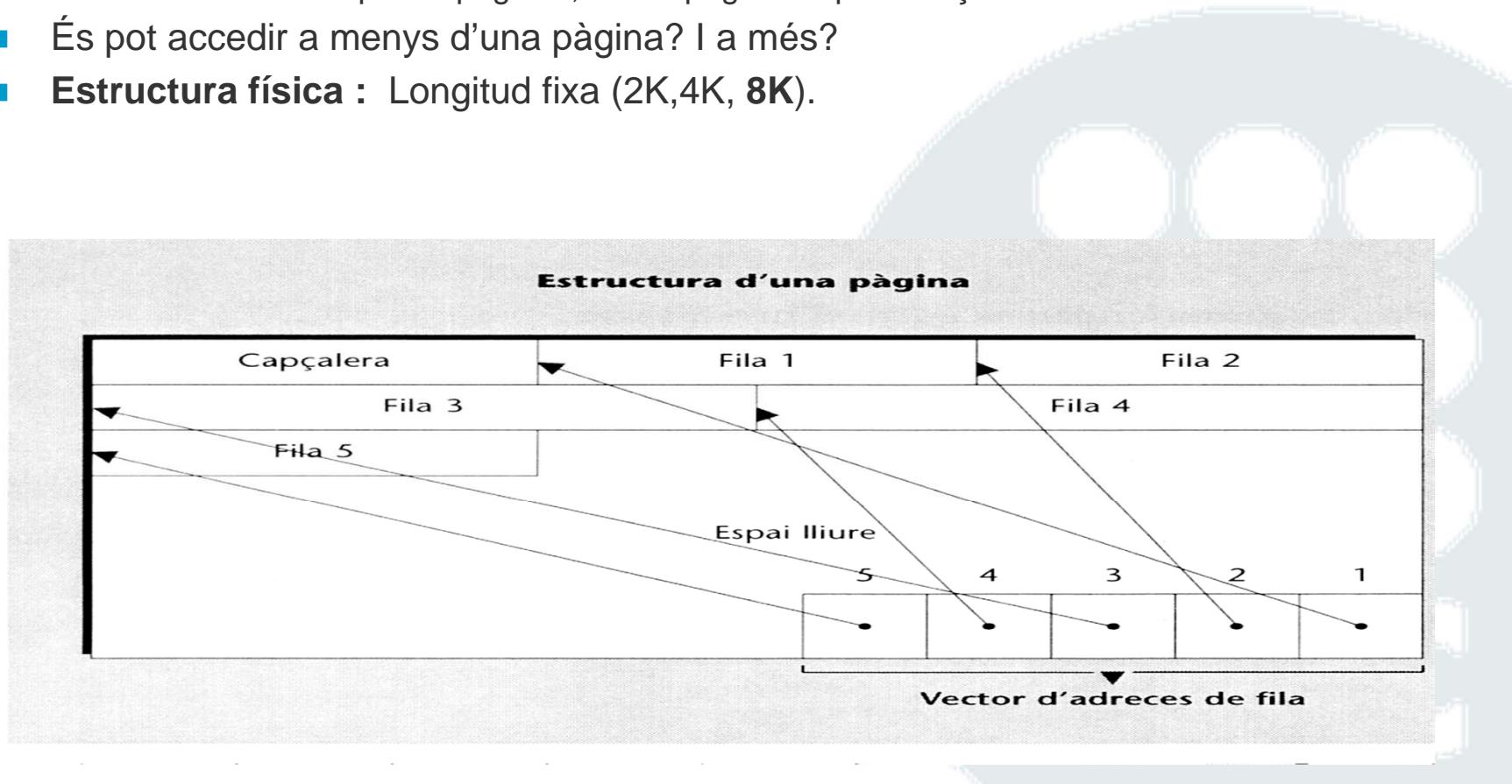
## El nivell físic

- Les dades s'emmagatzemen a discs magnètics controlats pels SO, que és qui realment efectua les lectures i escriptures. Ara bé, és l'SGBD el que decideix quan fer aquestes operacions i el que coneix com estan físicament estructurades les dades i les pot interpretar.
- Hi ha tres components essencials:
  - Fitxer: és la unitat global a partir de la qual el sistema operatiu gestiona les dades en els discs magnètics
  - Extensió (extent): és la unitat d'adquisició d'espai per a cada fitxer. L'extensió és un múltiple enter de la pàgina.
  - Pàgina (page): és el component físic més petit. Conté i emmagatzema les dades del nivell lògic



## La pàgina

- El concepte de pàgina en BD es pot veure de dos punts de vista:
  - **Unitat discreta de transport de dades (d'E/S)** entre la memòria externa (disc) i memòria interna. En SO normalment *bloc*.
  - **Unitat d'organització de les dades emmagatzemades.** L'espai del disc s'estructura en un nombre múltiple de pàgines, i cada pàgina es pot adreçar individualment.
- És pot accedir a menys d'una pàgina? I a més?
- **Estructura física :** Longitud fixa (2K,4K, 8K).



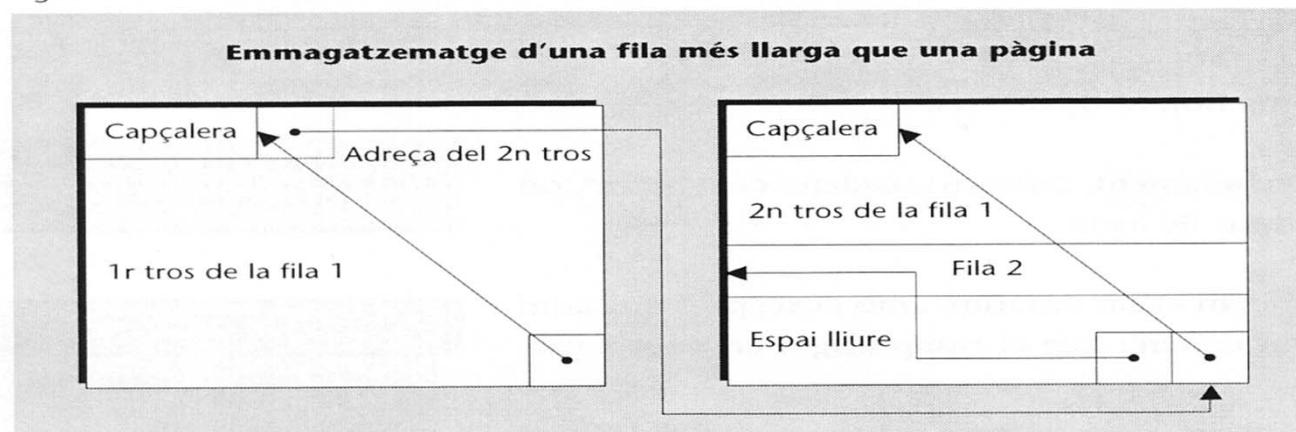
## La fila

- Estructura fila:



- La capçalera conté informació com: longitud total de la fila, identificador de la taula a la qual pertany.
- Veiem com seria l'emmagatzematge d'una fila més llarga que una pàgina, tot i que no és aconsellable tenir aquest tipus de files.

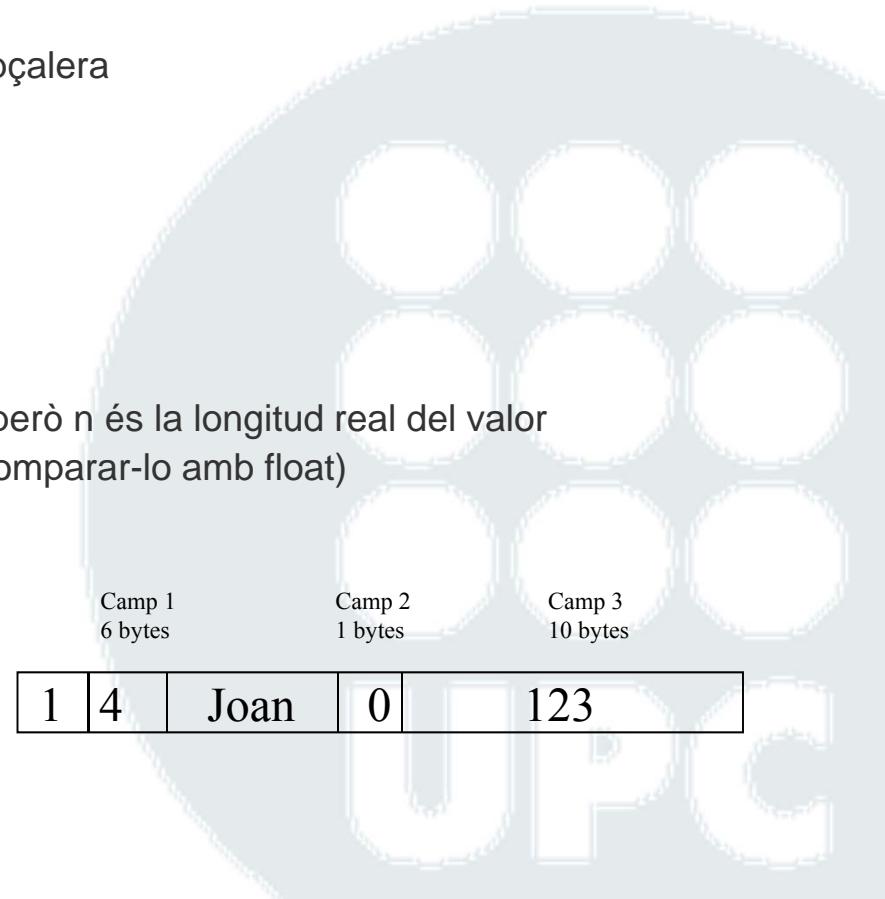
Figura 4



## El camp

- Estructura física:

Capçalera	Contingut
-----------	-----------
- A la capçalera del camp:
  - Ens diu si el camp és Null / not null (si el camp admet valors nuls)
  - Longitud del camp
  - Si el camp és de long fixa i not null no té capçalera
- Contingut. Formats més usuals:
  - Smallint: binari enter 2 bytes
  - Integer: binari enter 4 bytes
  - Float: coma flotant 8 bytes
  - Char(n): n bytes en ascii
  - Char varying (varchar): n bytes en ascii però n és la longitud real del valor
  - Decimal(p,s):  $\lceil (p+1)/2 \rceil$  (és interessant comparar-lo amb float)
  - Date: aaaa | mm | dd 4 bytes
- Exemple
  - Create table T ( camp1 varchar(10),  
camp2 varchar(10),  
camp3 char(10) not null)
  - Insert into T values ("Joan", NULL, "123")



## Motivació Emmgatzematge 3

```
drop table bulto;  
Create table bulto(a integer primary key, b varchar(60), c integer);
```

```
-- insertar 400000 tuples  
CREATE OR REPLACE FUNCTION ins_400000() RETURNS void  
AS $$  
DECLARE  
i INTEGER;  
BEGIN  
FOR i in 1 .. 400000 LOOP  
insert into bulto values(i,i,i);  
END LOOP;  
END;  
$$LANGUAGE plpgsql;
```

```
select * from ins_400000();
```

Explain  
Select \* from bulto where c=240000;

```
drop index prova;  
create unique index prova on bulto(c) with (fillfactor=80);
```

Explain  
Select \* from bulto where c=240000;

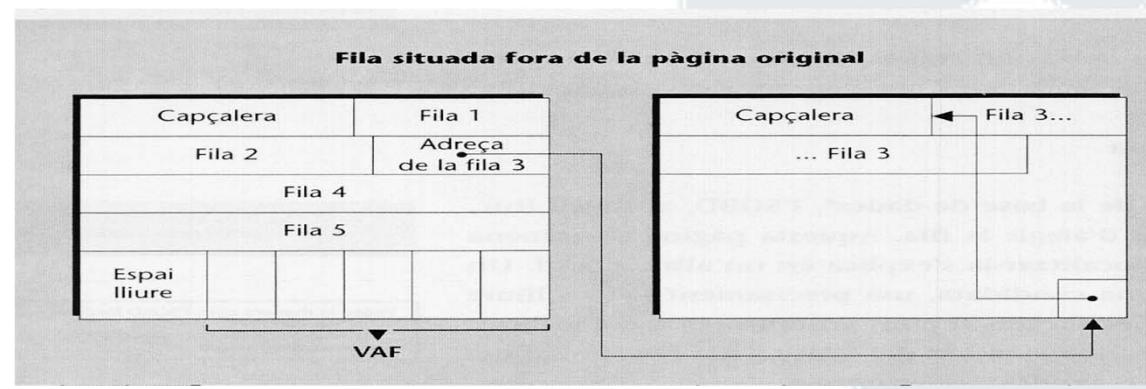
QUERY PLAN text	
1	Seq Scan on bulto (cost=0.00..7163.00 rows=1 width=14)
2	Filter: (c = 240000)

Sortida de dades	
QUERY PLAN text	
1	Index Scan using indx on bulto (cost=0.00..8.76 rows=1 width=69)
2	Index Cond: (c = 240000)

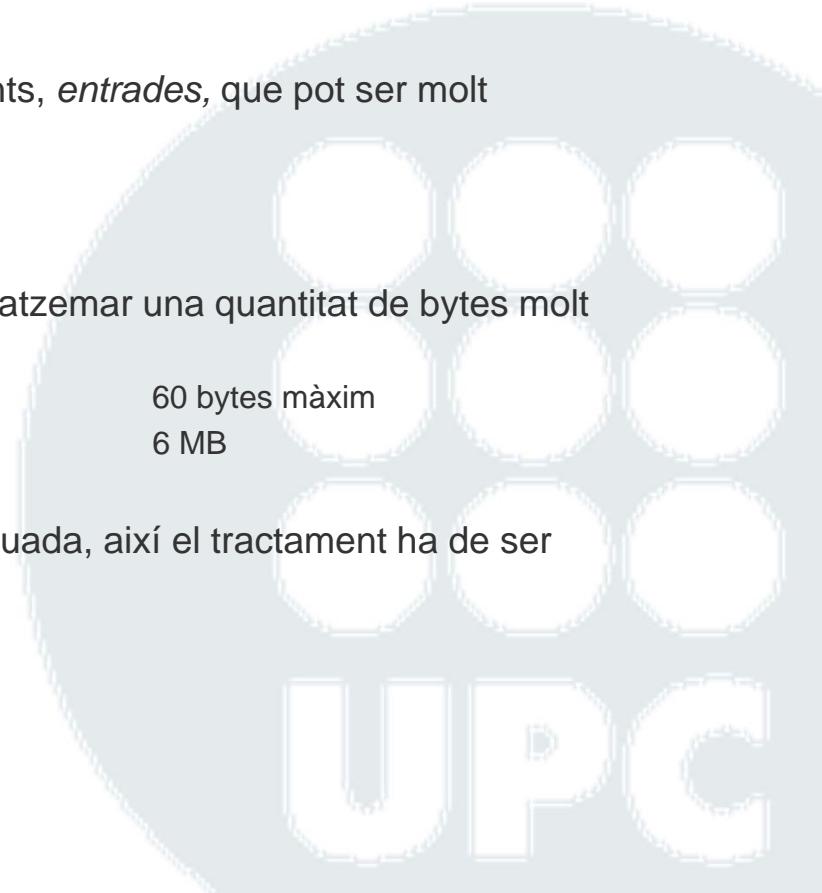
## Gestió de la pàgina

- Formatació
- Càrrega inicial de files
  - La primera darrere la capçalera i el vaf que l'apunta al final
  - La segona darrere la primera i el vaf just abans del darrer
  - Hem de deixar un % d'espai lliure
- Alta posterior d'una nova fila
  - Localitzar la pàgina candidata
  - Usar l'espai lliure; si ple, hi ha diverses opcions en funció del tipus d'espai en el que s'emmagatzemi (pàgina següent, punters, nova candidata)
- Baixa d'una fila
  - Alliberar l'espai ocupat
  - Reorganització interna de la pàgina (en funció de la política d'espai lliure)
- Canvi de longitud
  - Si disminució aleshores procés similar a la baixa
  - Si augment i espai suficient aleshores desplaçament de les files que segueixen
  - Si augment i no espai



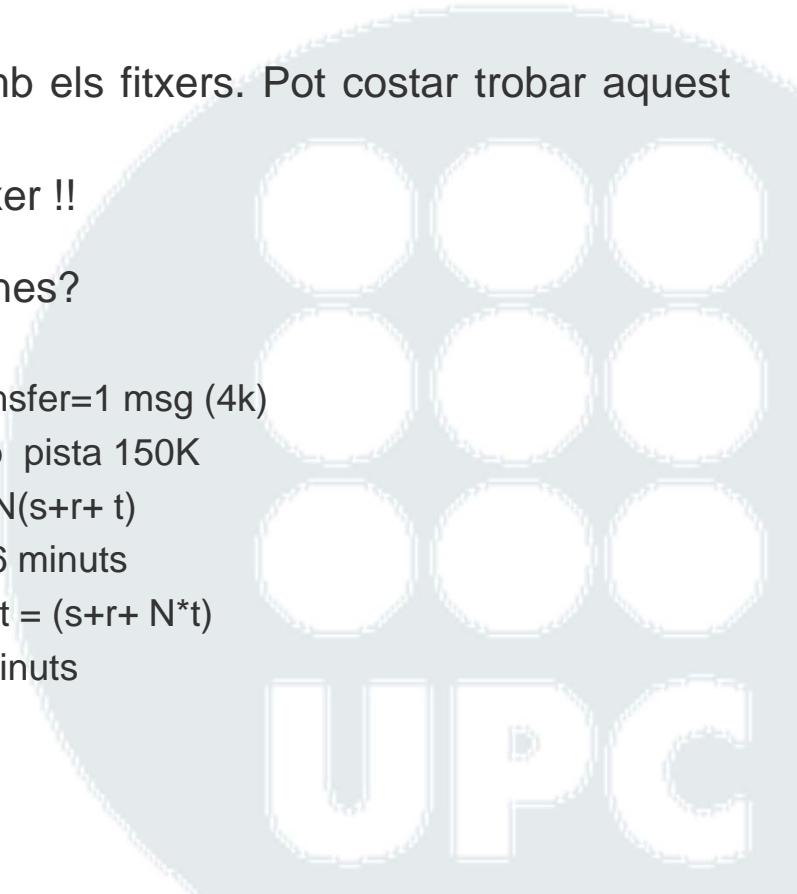
# Altres tipus de pàgines

- Pàgines d'altres components: vistes, disparadors, procediments
    - Totes les definicions al catàleg → com les taules
  - Pàgines d'índexs
    - Un índex és un conjunt d'enregistraments, *entrades*, que pot ser molt voluminós → com les taules
  - Pàgines d'objecte gran
    - La representació física obliga a emmagatzemar una quantitat de bytes molt superior a la de les dades tradicionals:
      - Nom clients                    varchar(60)                    60 bytes màxim
      - Powerpoint                    transparències                6 MB
    - La representació tradicional no és adequada, així el tractament ha de ser diferent → més endavant el veurem



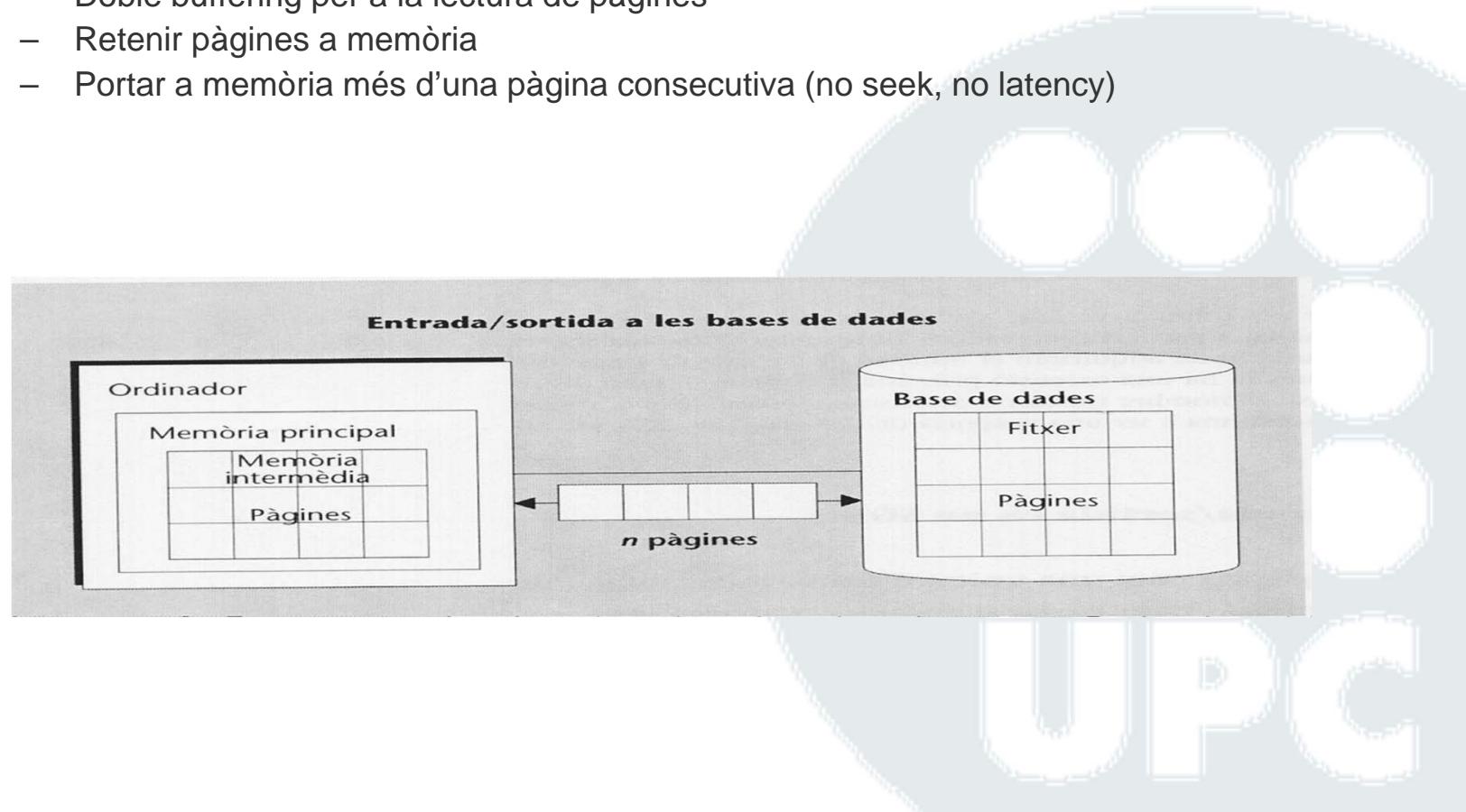
## Extensió i Fitxer

- Una extensió és un nombre enter de pàgines consecutives que el SO adquireix a petició de l'SGBD quan aquest detecta que necessita més espai per a un fitxer determinat. Automàtica.
- Un fitxer és un conjunt d'extensions.
- L'SGBD interacciona amb el SO i no amb els fitxers. Pot costar trobar aquest terme de manera explícita.  
A alguns SGBD petits tota la BD és un fitxer !!
- Per què han de ser consecutives les pàgines?
  - Eficiència dels tractaments seqüencials !!!
  - Disc: seek=12 msg; latència= 3 msg; Transfer=1 msg (4k)  
18Gb 9801 cilindres Cilindre 2Mb pista 150K
    - N pàgines no consecutives → cost =  $N(s+r+t)$   
»  $100000 (12 + 3 + 1) = 26'6$  minuts
    - N pàgines consecutives → aprox cost =  $(s+r+N*t)$   
»  $(12 + 3 + 100000) = 1,6$  minuts

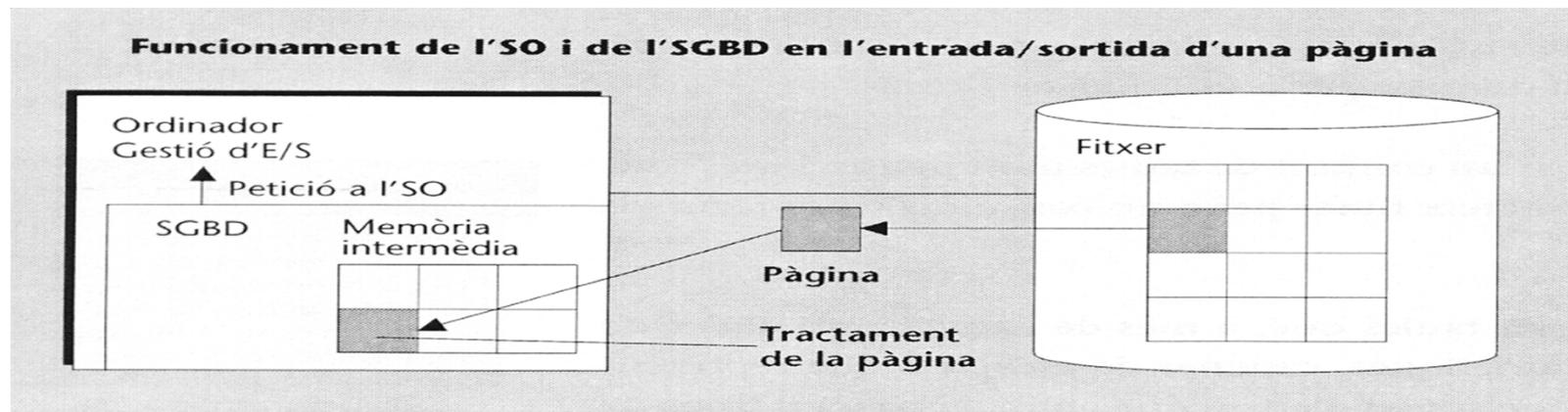


## Entrades/sortides en SGBD

- En BD la longitud de la pàgina és fixa per a tots els fitxers.  
Senzillesa: Rendiment
- Optimització E/S
  - Doble buffering per a la lectura de pàgines
  - Retenir pàgines a memòria
  - Portar a memòria més d'una pàgina consecutiva (no seek, no latency)



## Entrades/sortides en SGBD



- Hi ha diversos motius pels quals la gestió de les pàgines no es pot deixar en mans del sistema operatiu:
  - Degut al concepte de transacció, que només coneix l'SGBD, algunes pàgines s'han de gravar al disc en cert moments del temps: en el moment de commit, en el moment d'abort, etc
  - Optimització

## EL nivell virtual: Justificació i definició

- Justificació:

Si Taula  $\Leftrightarrow$  Fitxer, no cal un nivell intermedi que faci corresponder components lògics amb físics, però

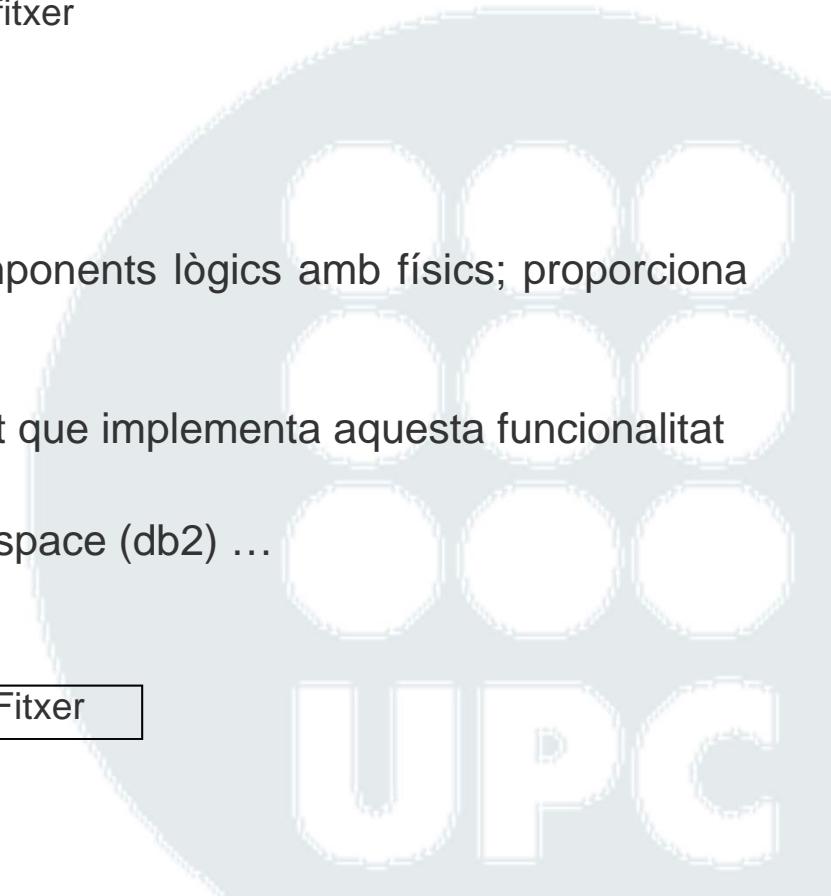
- Si taules molt grans: hi ha fragments en dispositius diferents
- Si taules molt petites: s'han d'agrupar en un fitxer
- Si hi ha objectes grans
- Si hi ha índexs, disparadors ...

- Definició:

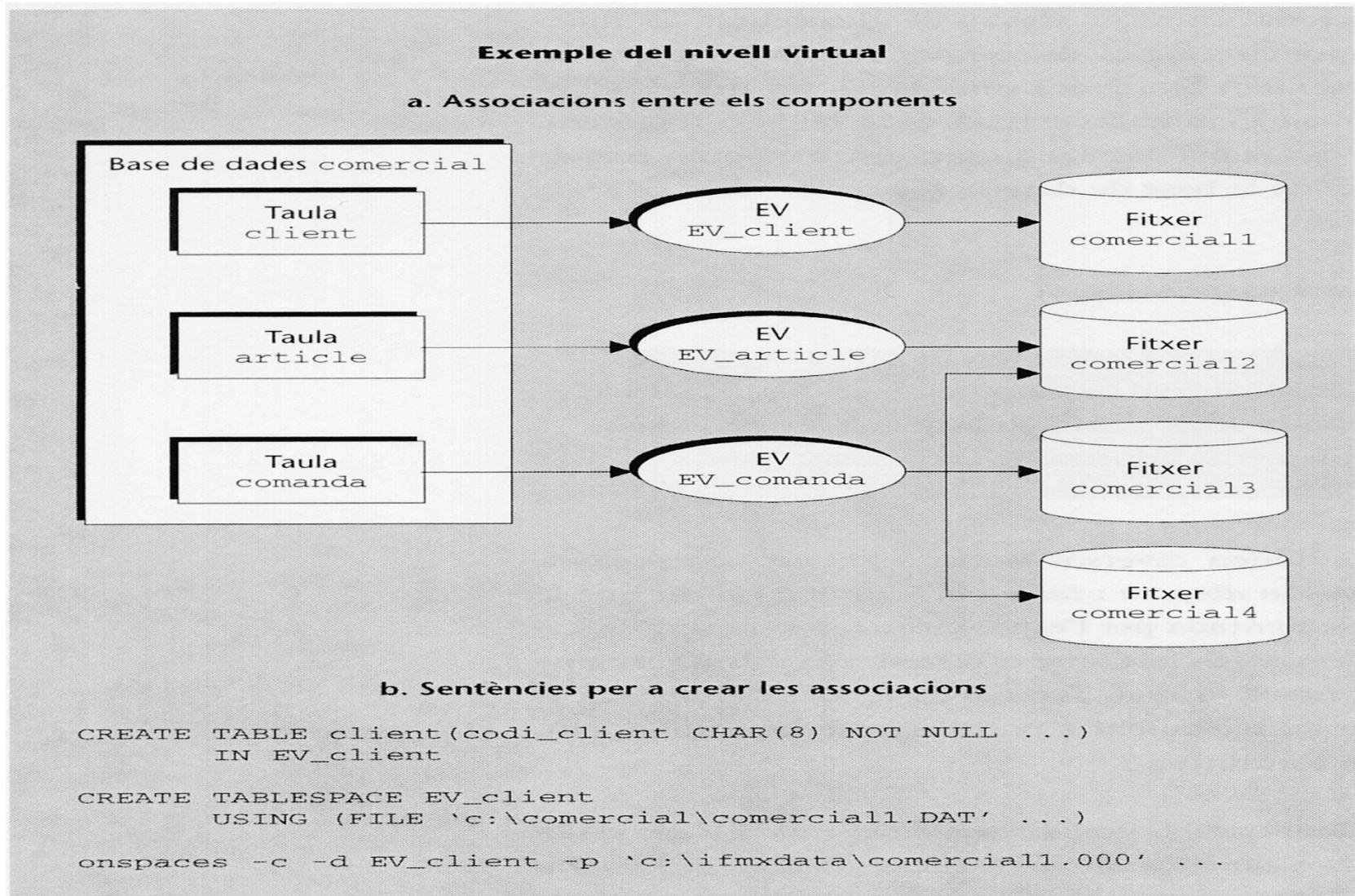
Nivell intermedi que permet relacionar components lògics amb físics; proporciona independència entre nivells. Flexibilitat

S'anomena Espai Virtual (EV) el component que implementa aquesta funcionalitat

- Noms comercials: dbspace (informix); tablespace (db2) ...
- Normalment,

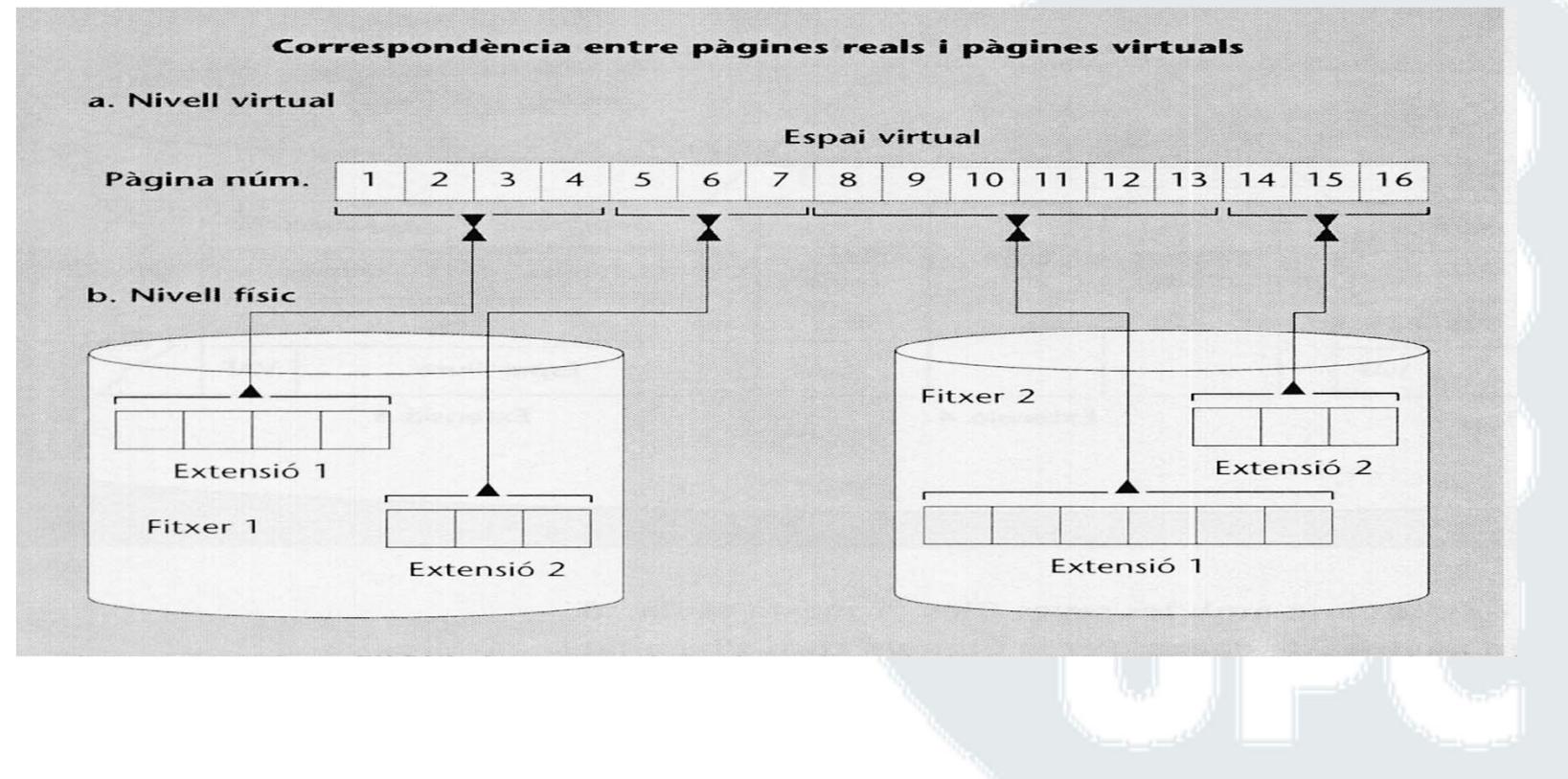


## El nivell virtual: Exemple

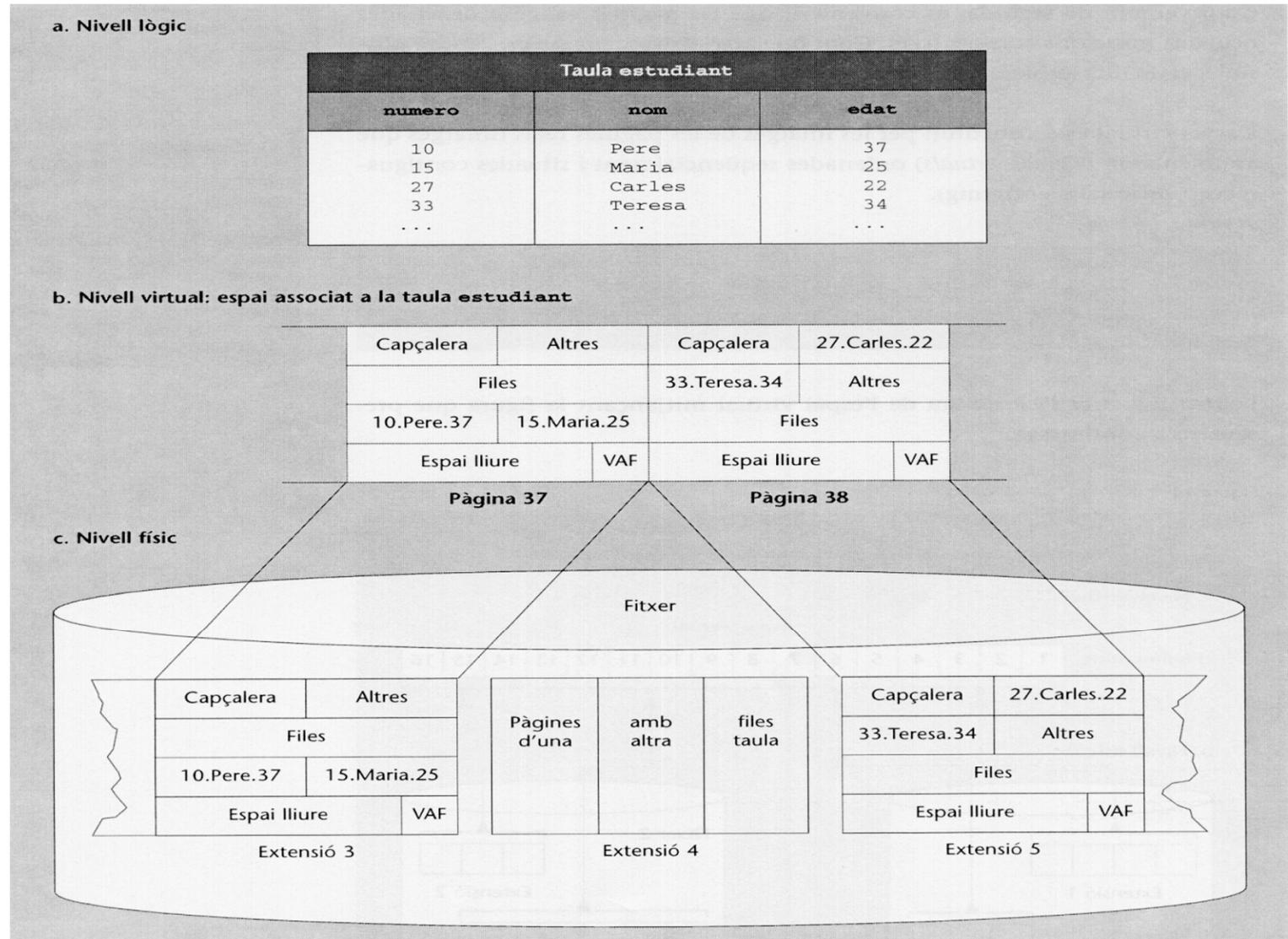


## Estructura de l'espai virtual

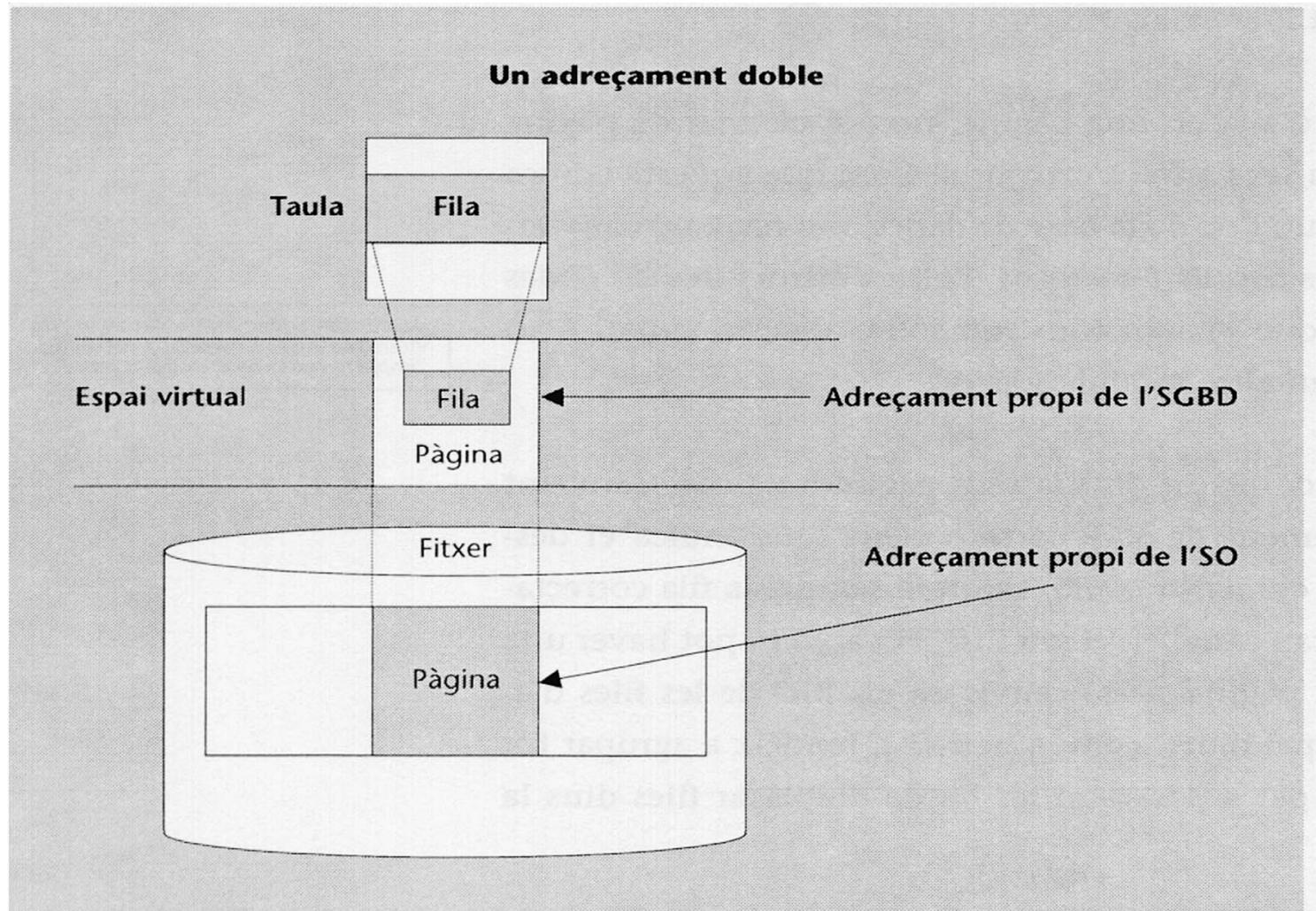
- Espai virtual: visió diferent de les pàgines físiques, sense duplicar
  - Paral·lelisme: Vistes(taules) Memòria virtual(real)
- Espai virtual: Seqüència de pàgines virtuals que es relacionen una a una amb les reals del nivell físic



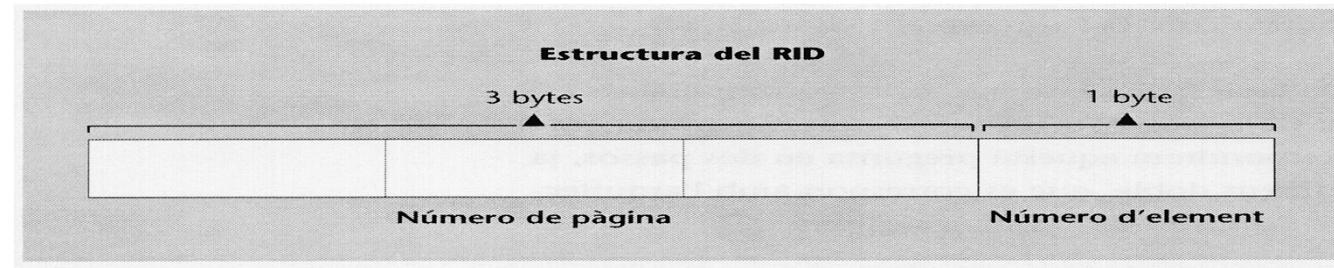
## Estructura de l'espai virtual: Exemple dels tres nivells



## Adreçament en un SGBD



## Record IDentifier (RID)



- Una fila està clavada (pinned) a les pàgines, fins i tot, si creix i no hi cap !!
- Una fila es pot moure dintre la pàgina
- Això ens dóna independència del dispositiu concret

## Tipus d'espais virtuals

### ■ Espai de taules

- Per a taules no molt grans, ni lligades a altres taules
- Seleccions eficients, però joins no

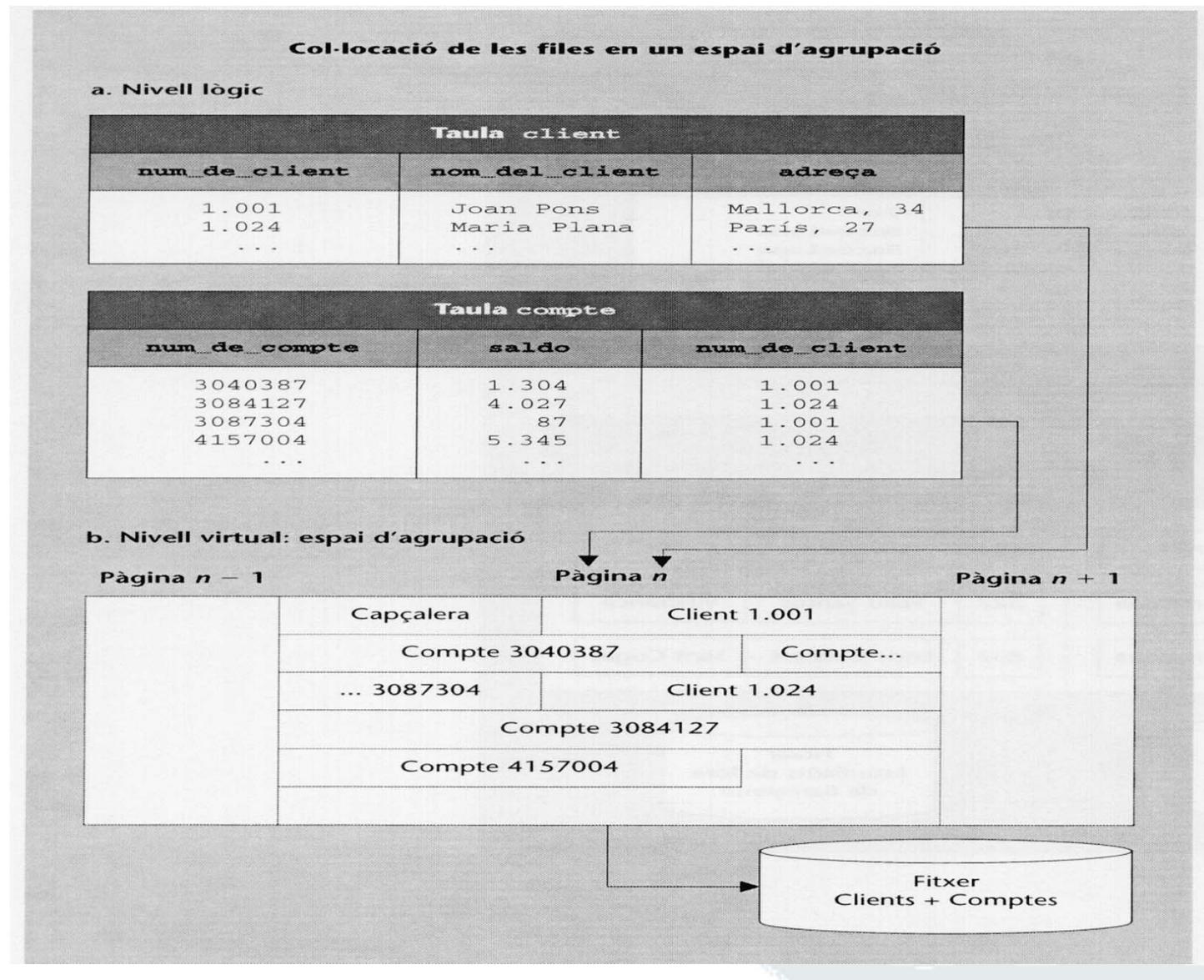


### ■ Espai d'agrupació

- Per a taules molt relacionades on l'accés quasi sempre és conjunt
- Join eficient, però la selecció parcial no



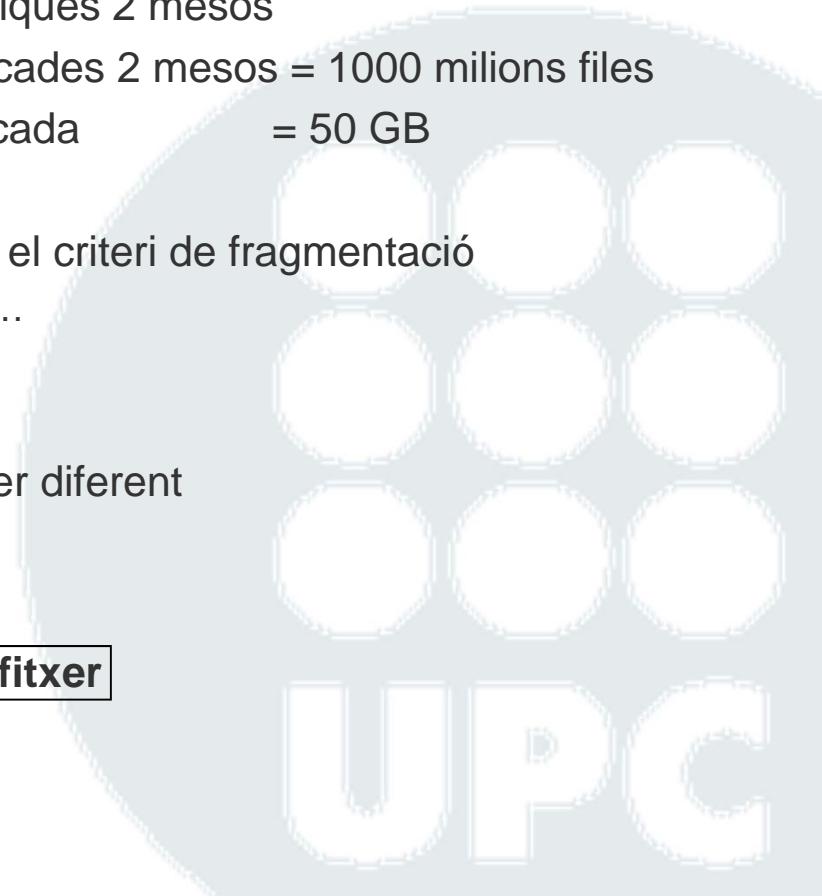
## Tipus d'espais virtuals: Exemple d'espai virtual d'agrupació



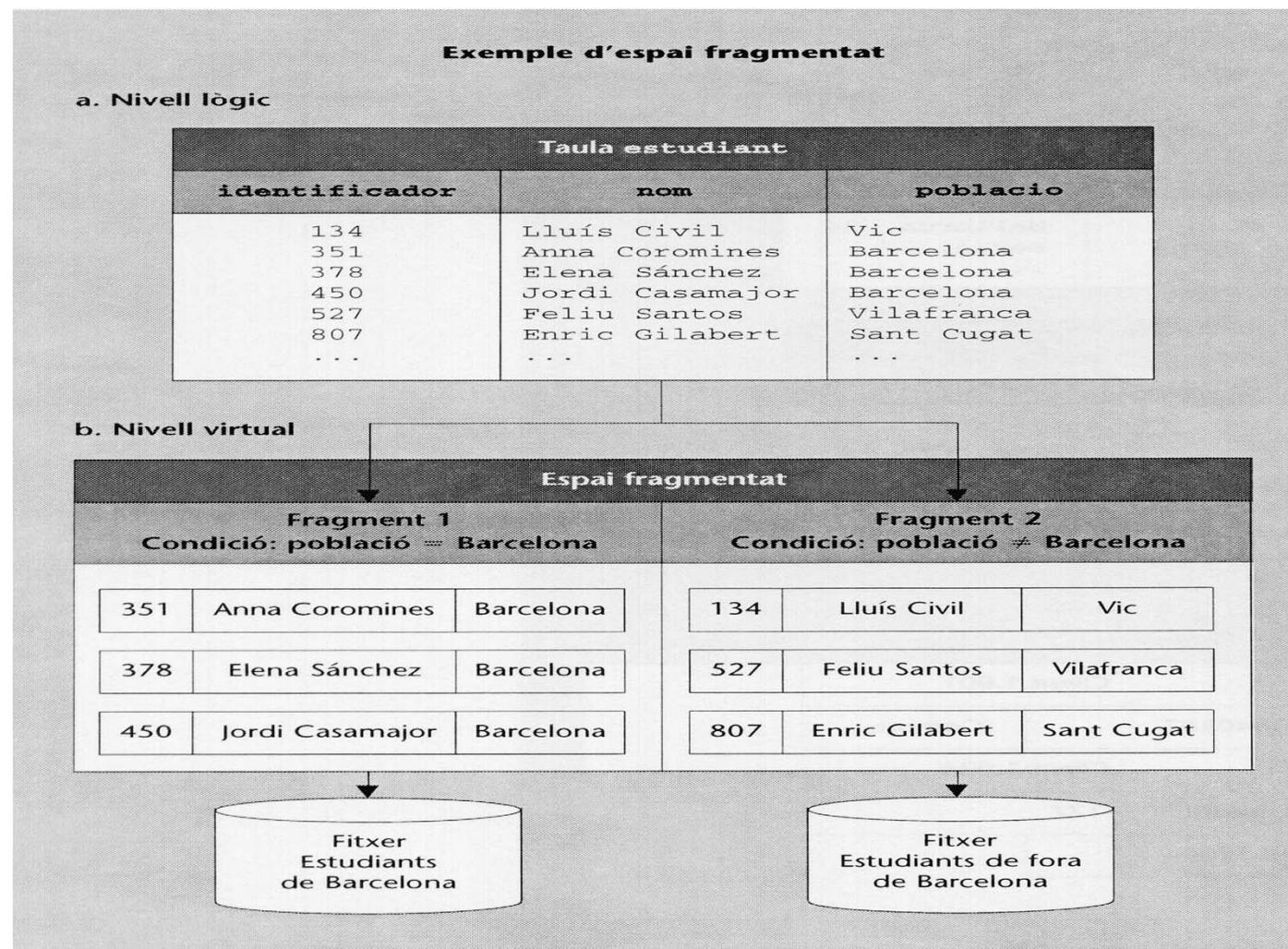
## Tipus d'espais virtuals

### ■ Espai fragmentat

- Per a taules molt grans
  - » Ex: Info. Trucades telefòniques 2 mesos
  - » 10 milion abonats 100 trucades 2 mesos = 1000 millions files
  - » 1000 millions 50 bytes trucada = 50 GB
- La taula s'associa a l'espai i cal establir el criteri de fragmentació
  - Ex: valor d'un camp clients < 20.000 ...
  - Aleatoriament i uniforme (Round robin)
- Es pot associar cada fragment a un fitxer diferent
  - Optimització



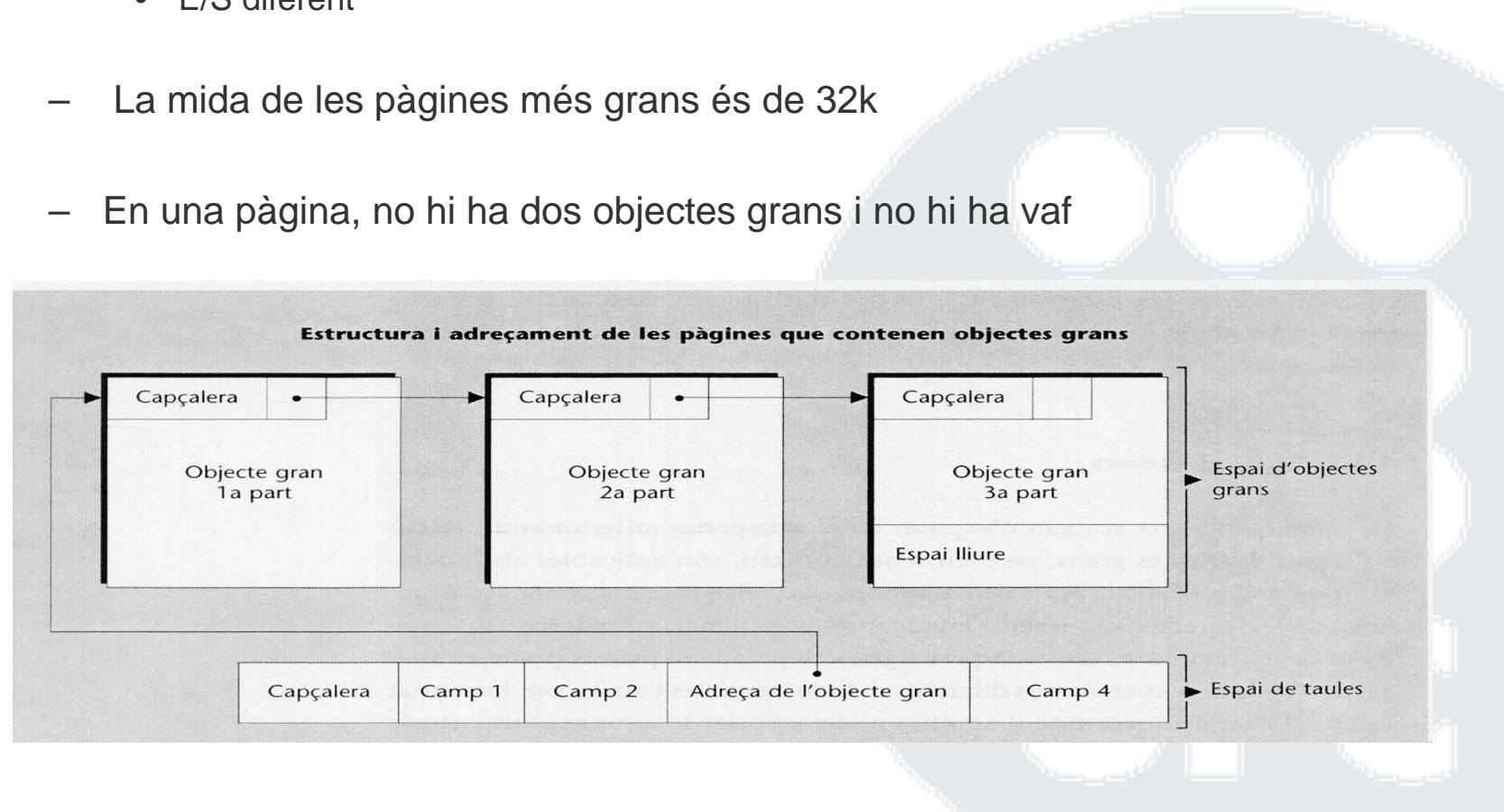
## Tipus d'espais virtuals: Exemple espai virtual fragmentat



## Tipus d'espais virtuals

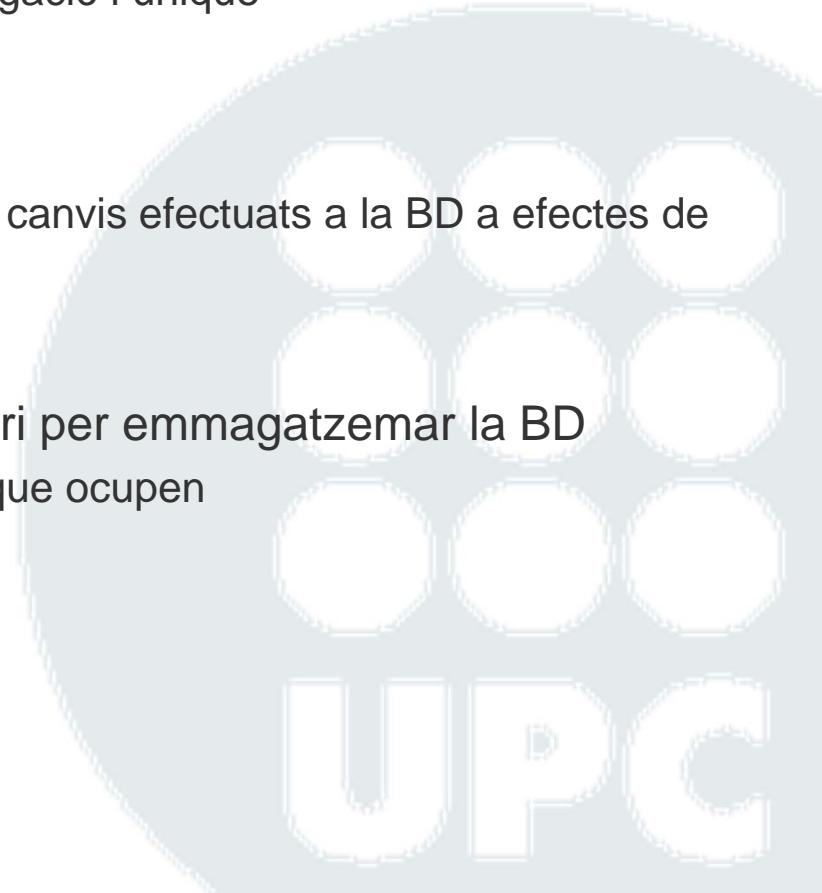
### ■ Espai d'objectes grans

- Els objectes grans s'emmagatzemen separadament
  - L'accés als tipus tradicionals és més freqüent
  - E/S diferent
- La mida de les pàgines més grans és de 32k
- En una pàgina, no hi ha dos objectes grans i no hi ha vaf



## Altres aspectes

- Catàleg
  - Normalment, és un EV de taules
- Taules temporals
  - Select into, group by, funcions d'agregació i unique
  - Normalment, és un EV de taules
- Dietari
  - Fitxer(s) que guarden les còpies dels canvis efectuats a la BD a efectes de restauració
  - Normalment, és un EV de taules
- L'ABD ha de calcular l'espai necessari per emmagatzemar la BD
  - Evitar un càlcul erroni de files\*bytes que ocupen
  - A més, cal tenir en compte:
    - Catàleg
    - Taules temporals
    - Dietari
    - Miralls
  - Mirar manuals!



## Implementació de mètodes d'accés

- Introducció
- Objectius
- Mètodes d'accés a una base de dades
  - Per posició
  - Per valor
  - Per diversos valors
- Implementació dels accessos per posició.
- Implementació dels accessos per valor
  - Característiques generals dels índex
  - Arbre B+
  - Índexs agrupats
- Implementació dels accessos per diversos valors



## Implementació de mètodes d'accés: Introducció

- Sempre que es llegeix o s'actualitza alguna dada d'una base de dades es fa mitjançant algun dels mètodes d'accés disponibles en un SGBD.
- Els SGBD estructuren les seves dades en pàgines dels espais virtuals i aquests físicament s'emmagatzemen en pàgines físiques dels discs magnètics. De manera que una lectura o actualització a nivell lògic implica:
  - un conjunt de lectures o actualitzacions de pàgines del nivell virtual.
  - un conjunt de lectures o actualitzacions de pàgines del nivell físic.
- Estudiarem únicament el cas d'accisos a dades emmagatzemades en una única taula situada en un únic espai virtual, encara que el que s'estudia és directament aplicable a accessos a diverses taules si les tenim en un únic espai d'agrupació.
- Implementar un mètode d'accés d'una o altra manera pot comportar un cost més alt o més baix. Establirem com a convenció per a l'estimació del costos:
  - Considerem només el cost d'entrades/sortides, no de CPU
  - Simplificació: Compten només el nombre de pàgines que es llegeixen o escriuen
    - NO: la pàgina està a memòria; no seek no latency

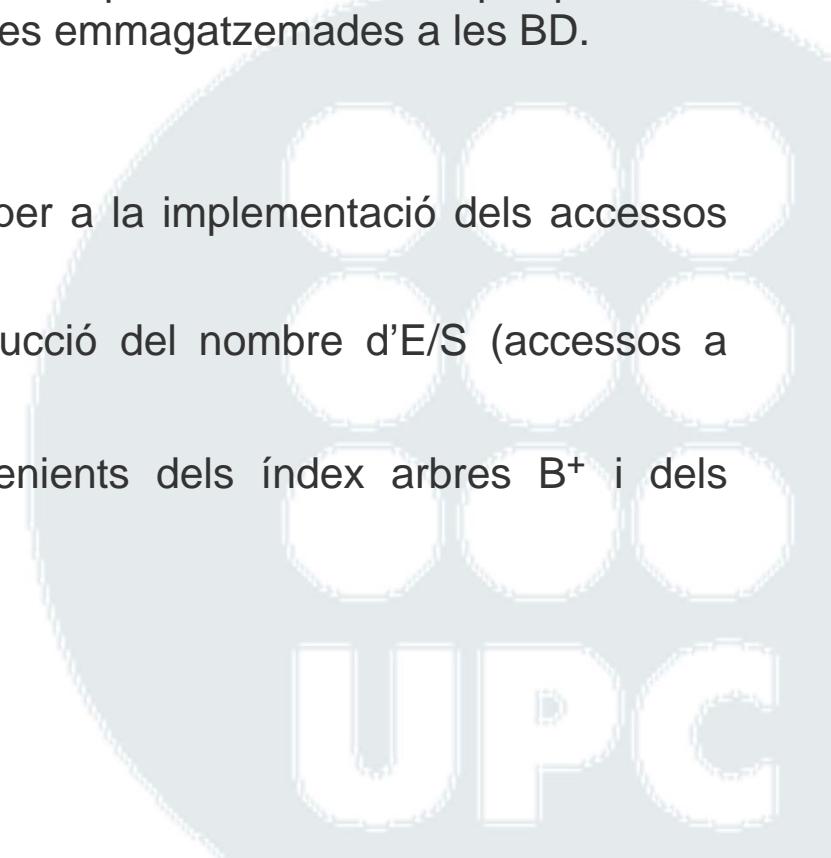
## Implementació als mètodes d'accés: Objectius

### ■ Objectius Generals

- Conèixer els diferents mètodes d'accés que són necessaris per poder fer consultes i actualitzacions a les dades emmagatzemades a les BD.

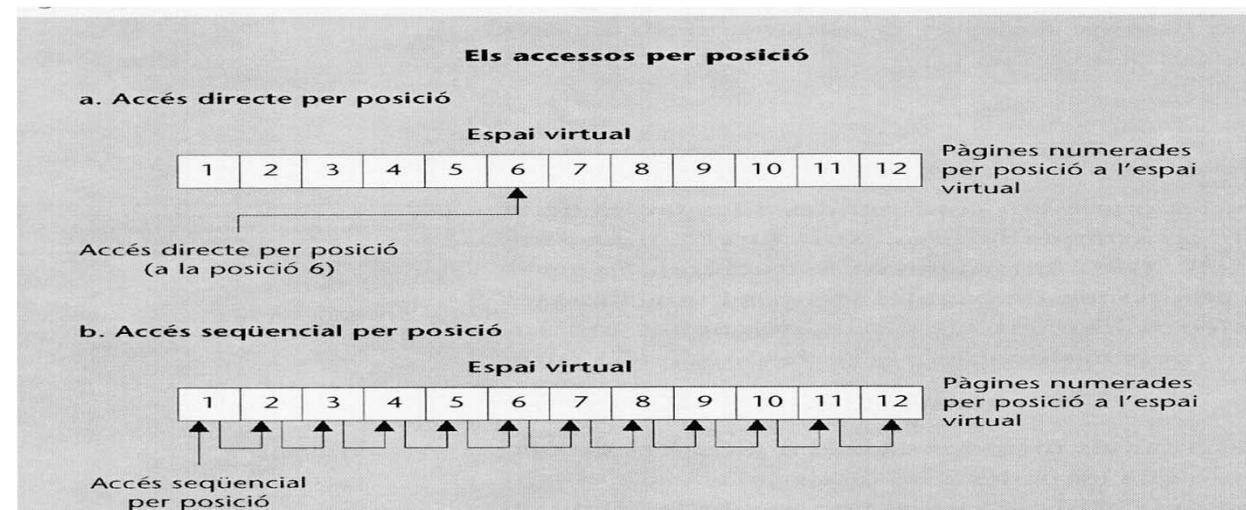
### ■ Objectius Concrets

- Comprendre la utilitat dels índexs per a la implementació dels accessos per valor.
- Entendre la importància de la reducció del nombre d'E/S (accessos a disc) en les implementacions.
- Entendre els avantatges i inconvenients dels índex arbres B<sup>+</sup> i dels agrupats per fer accessos per valor.



## Mètodes d'accés: Accés per posició

- **Directe:** Obtenir una pàgina que té un número de pàgina determinat.
- **Seqüencial:** Obtenir les pàgines d'un espai seguint l'ordre definit pels seus números de pàgina.



- Suficient per casos simples:

Empleats(nemp, nom, despatx, sou)

**Directe:** Insert into Empleats values (25, 'Joan', 150, 200k)

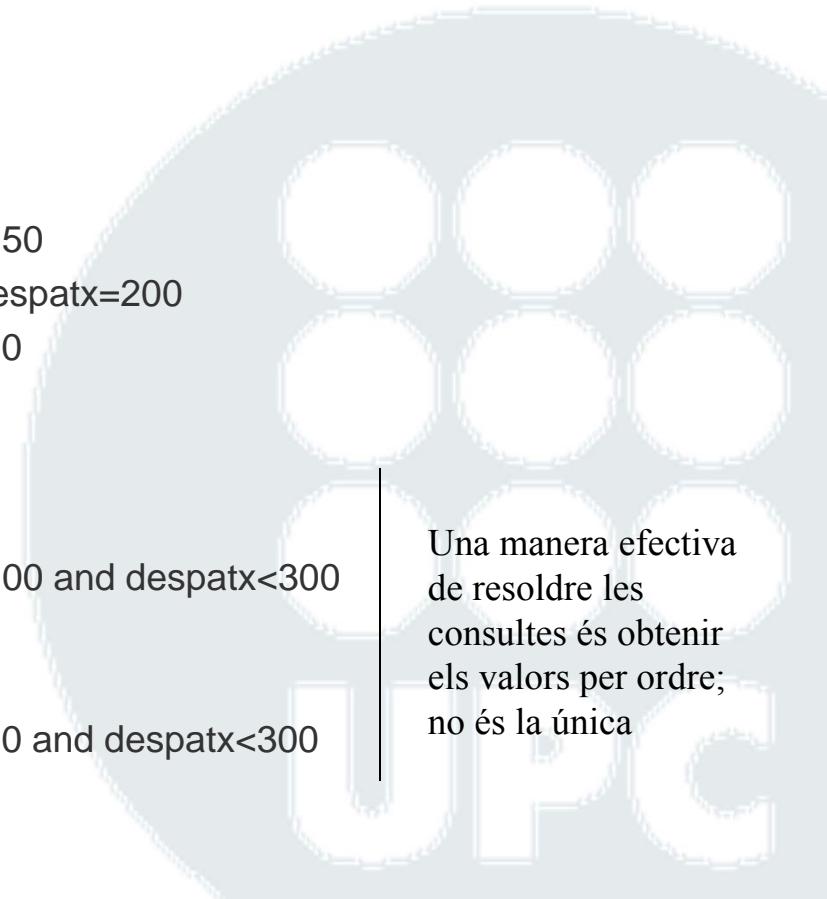
**Seqüencial:** Select \* from Empleats

## Mètodes d'accés: Accés per valor

- **Directe:** Obtenir totes les files que contenen un valor determinat d'un atribut
- **Seqüencial:** Obtenir totes les files per l'ordre dels valors d'un atribut

### ■ Exemples

- Directe:
  - Select \* from Empleats where despatx=150
  - Update Empleats set sou=250k where despatx=200
  - Delete from Empleats where despatx=150
- Seqüencial
  - Select \* from Empleats order by despatx
  - Select \* from Empleats where despatx>100 and despatx<300
  - Update Empleats set sou=250k  
where despatx >100 and despatx<300
  - Delete from Empleats where despatx>100 and despatx<300



Una manera efectiva  
de resoldre les  
consultes és obtenir  
els valors per ordre;  
no és la única

## Mètodes d'accés: Accés per diversos valors

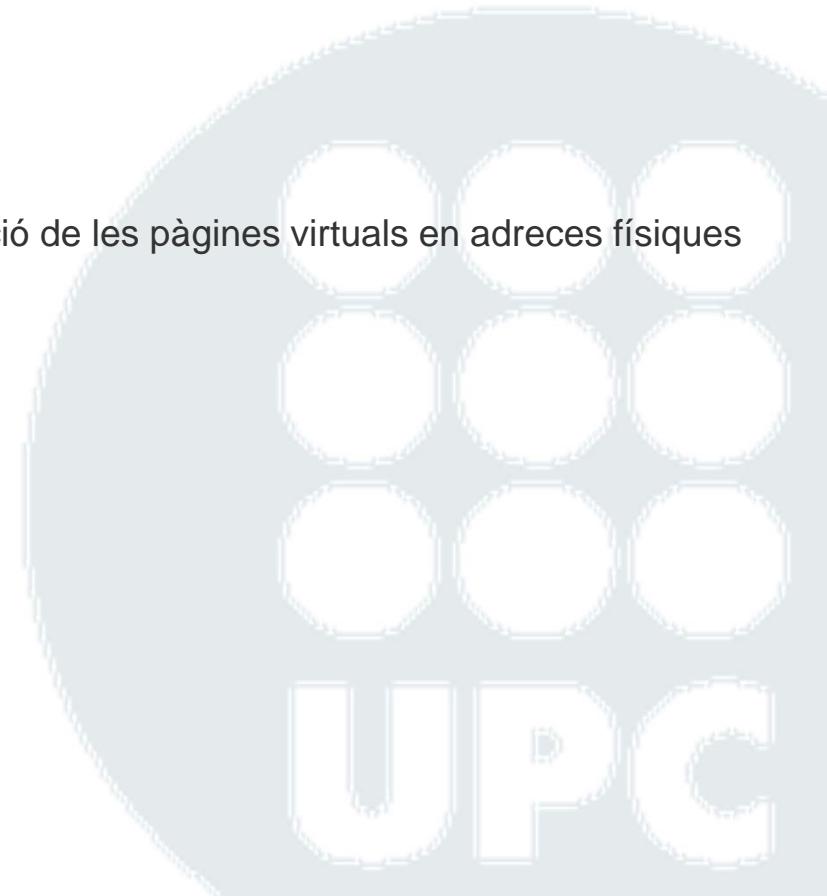
- Accedir a les files per valors de diversos atributs
- Es poden fer accessos directes o seqüencials

- Exemples
  - Select \* from Empleats order by despatx, sou  
Directe | seqüencial
  - Select \* from Empleats where despatx=150 and sou=200k  
Directe | seqüencial
  - Delete from Empleats where despatx>100 and sou>180k  
Directe | seqüencial
  - Select \* from Empleats where despatx=150 and sou>200k  
Directe | seqüencial
- Exemples d'accessos mixtos (seqüencial i directe) per diversos valors
  - Select \* from Empleats where sou=200 order by despatx
  - Delete \* from Empleats where despatx>100 and despatx<200 and sou=200k



## Implementació dels accessos per posició

- Els SGBD es basen en el sistema operatiu. Les rutines d'E/S del sistema operatiu permeten obtenir una pàgina física donada la seva adreça, i també permeten enregistrar una pàgina física concreta.
  
- Accés directe per posició
  - L'SGBD transforma els números de posició de les pàgines virtuals en adreces físiques
    - » Cost : 1 pàgina
- Accés seqüencial per posició
  - Similar
    - » Cost : N pàgines



## Implementació dels accessos per valor

- Per implementar de manera eficient l'accés per valor s'usa unes estructures de dades auxiliars anomenades **ÍNDEXS**
- Per què són necessaris? Cost de les solucions alternatives a l'ús d'índexs:

**Select \* from Empleats where despatx=150**

S e q ü e n c i a	accés seqüencial per posició	MIG	MAX
T a u l a	I d e m	N / 2	N
T a u l a      ord . físicament	Cerca dicotòmica Accés directe posició	$\log_2 N$	$\log_2 N + 1$
	Interpolació	$\log_2 \log_2 N$	$\rightarrow N$
T a u l a      ord . lògicament	Cadenacurtaillarga	$\sqrt{N}$	$2\sqrt{N}$

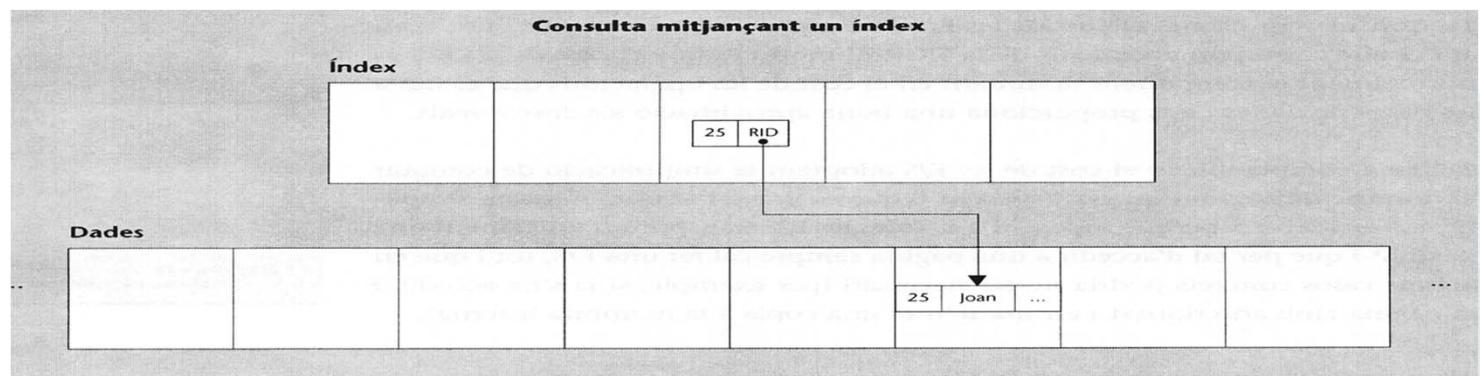
Els costos d'ambdues consultes pel cas de 300000, i 23M d'empleats seria:

- 300.000 empleats (registres), 10 registres/pàgina, N=30.000 pàgines  
Costos: 15.000; 15.000; **15; 4; 174** ordre físic !!
- 23.000.000 empleats (registres), 10 registres/pàgina, N=2.300.000 pàgines  
Costos: 1.150.000; 1.150.000; **22; 5; 1516** ordre físic !!

**Select \* from Empleats where sou < 200000**

## Característiques generals dels índexs

- Utilitat semblant als índexs de llibres
- Les cerques poden ser més ràpides que si es fan sobre les dades ja que els índexs ocupen molt menys espai que les dades
  - Els índexs contenen parelles (valor, RID) → **entrades**

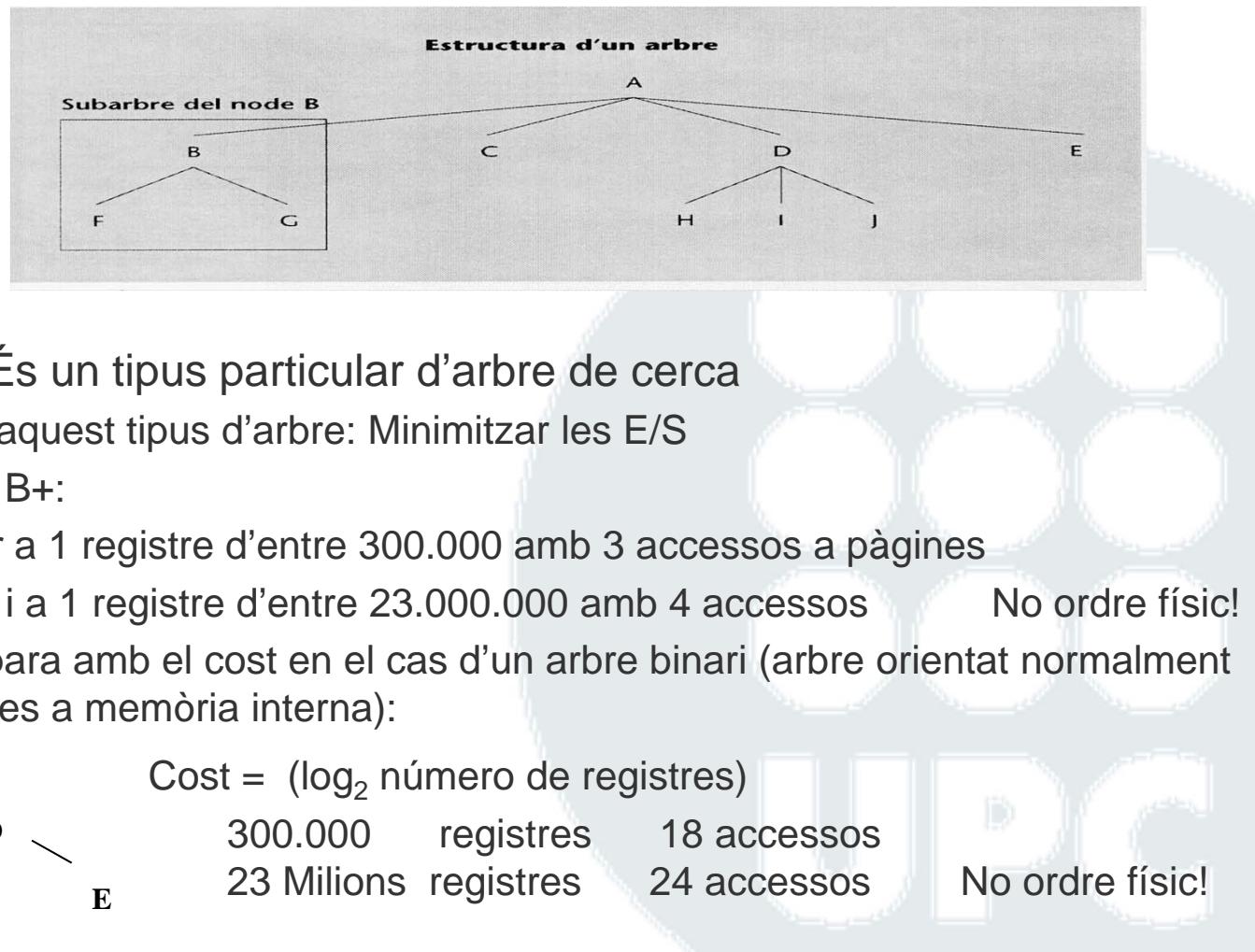


- Fan accés per valor via accés per posició!!!
- Diverses maneres d'estructurar els índexs:
  - Arbre B+ (accés per valor i seqüencial per valor)
  - Altres tipus (Funcions de dispersió, Bounded disorder files, R-tree,...)
- Una característica molt útil és usar múltiples índexs sobre una relació
  - Versus dades estructurades en forma d'índex

## Arbres B+ Introducció

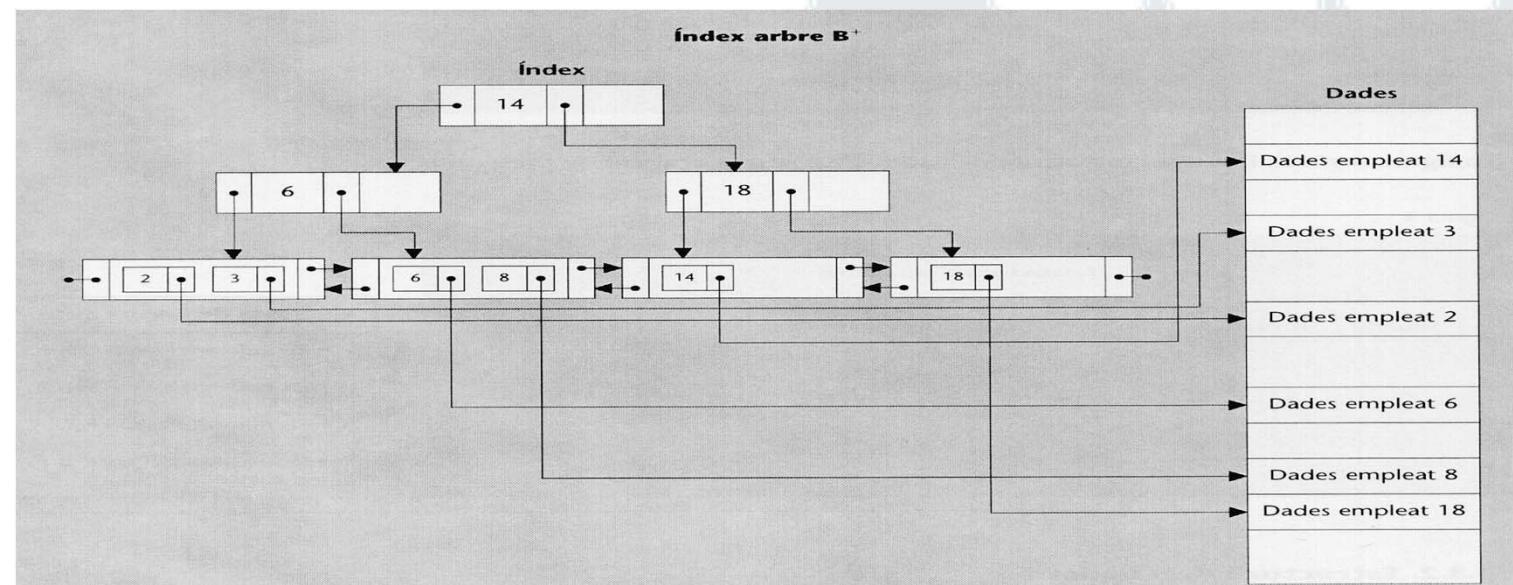
- Arbres:

- Nodes. Arrel. Fulles. Nodes interns. Subarbre.
- Nivell de l'arrel = 1; Nivell d'un node = nivell del pare + 1



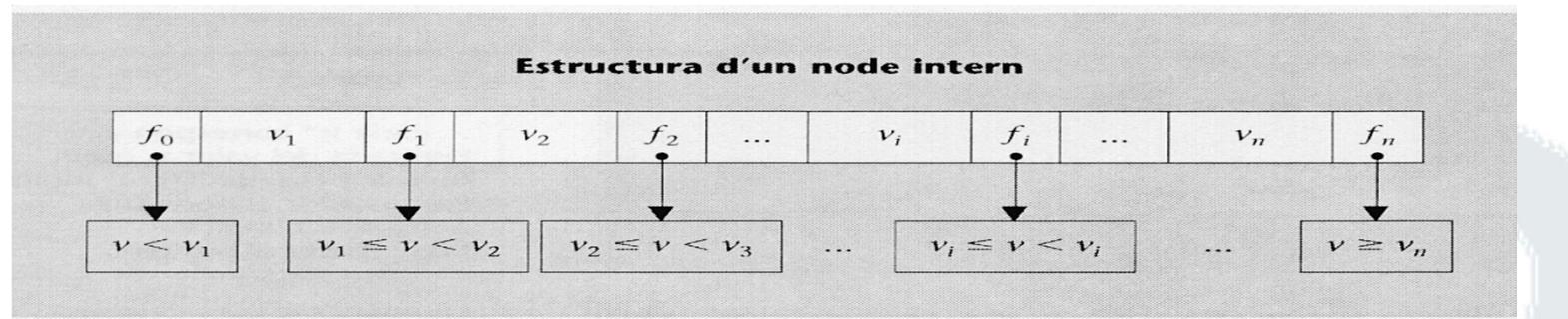
## Arbres B+ Estructura

- Tot arbre té un número d'ordre  $d$  que indica la capacitat dels nodes.
  - Arbre B<sup>+</sup> d'ordre  $d$ , els nodes contenen com a màxim  $2d$  valors
- Els nodes interns i els nodes fulla tenen estructures diferents:
  - Els nodes fulla són els que contenen totes les entrades del índexs (valor i RID)
  - Els nodes interns tenen com a objectiu dirigir la cerca !
    - No tenen entrades, només valor i apuntadors a altres nodes
  - Els nodes fulla estan connectats per apuntadors dobles



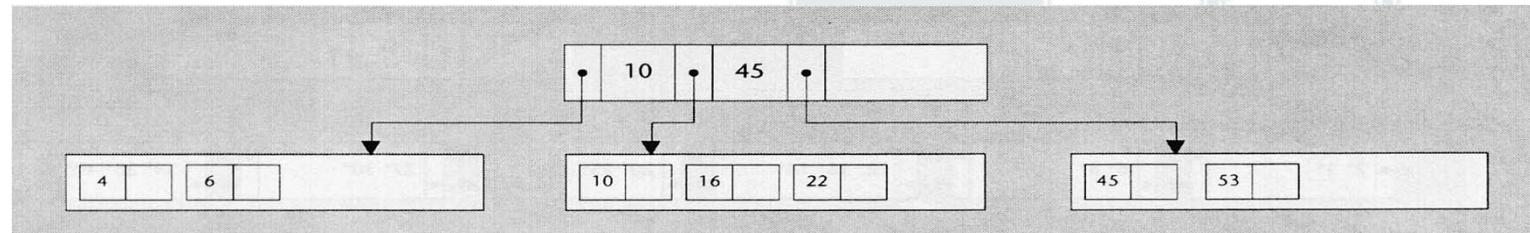
## Arbres B+ Estructura node intern

- Un node intern conté valors i apuntadors cap els seus nodes fills
- L'estructura d'un node intern que conté  $n$  valors, amb  $n \leq 2d$ :



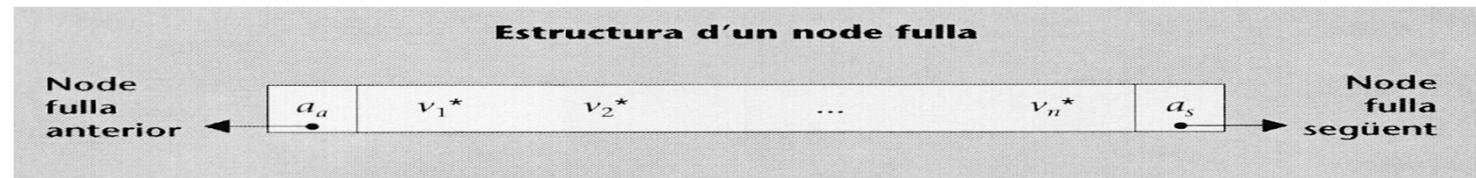
En el cas màxim:  $2d$  valors i  $2d+1$  apuntadors

- Exemple d'ordre 2:

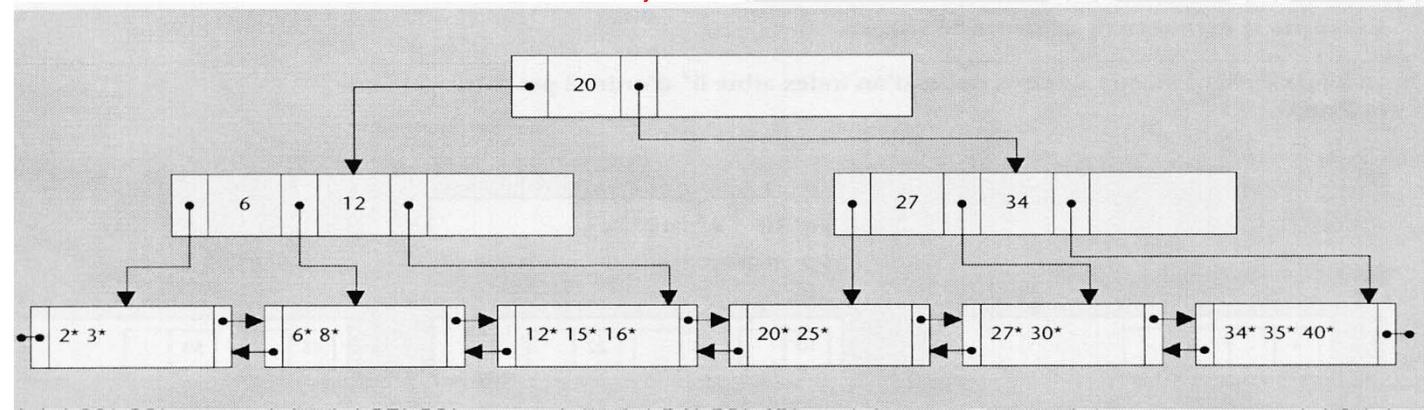


## Arbres B+ Estructura node fulla

- Els nodes fulla contenen:
  - Entrades formades per [valor, RID] que escriurem  $v^*$
  - Apuntadors a la fulla anterior i següent

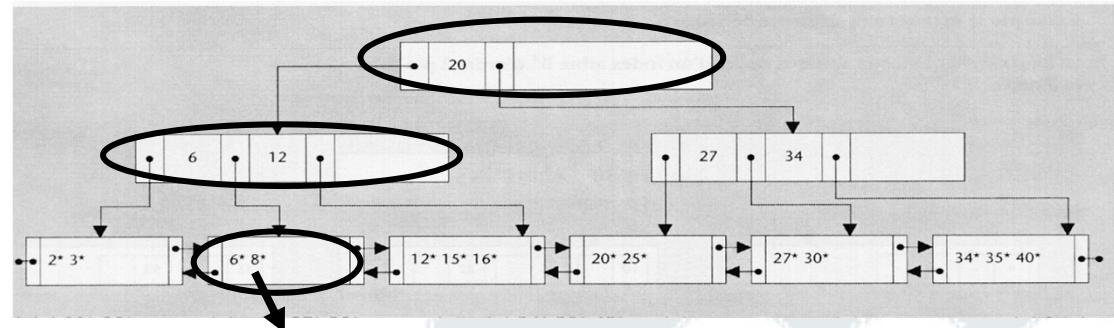


- Cas màxim:  $2d$  (valors+rid) + 2 apuntadors a fulla
- Condicions addicionals:
  - Tots els valors d'un node fulla són més petits que els valors de les fulles següents
  - Tots els valors d'un node intern estan repetits a les fulles. Les fulles tenen tots els valors.
  - **Els nodes interns no tenen valors repetits.**

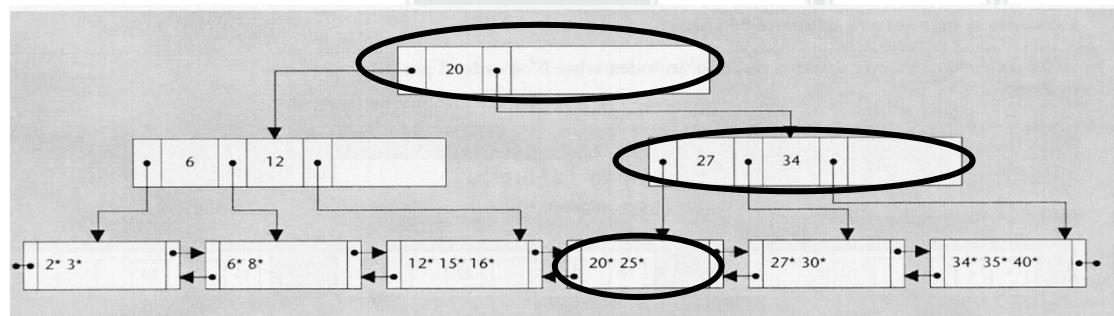


## Arbres B+ Accés directe per valor

- Per accedir al registre amb valor  $v$  cal
  - Localitzar la fulla que té l'entrada  $v$
  - Usar el RID de l'entrada per a trobar la fila cercada
- Exemples: Localitzar el registre amb el valor 8



Localitzar el registre amb el valor 26



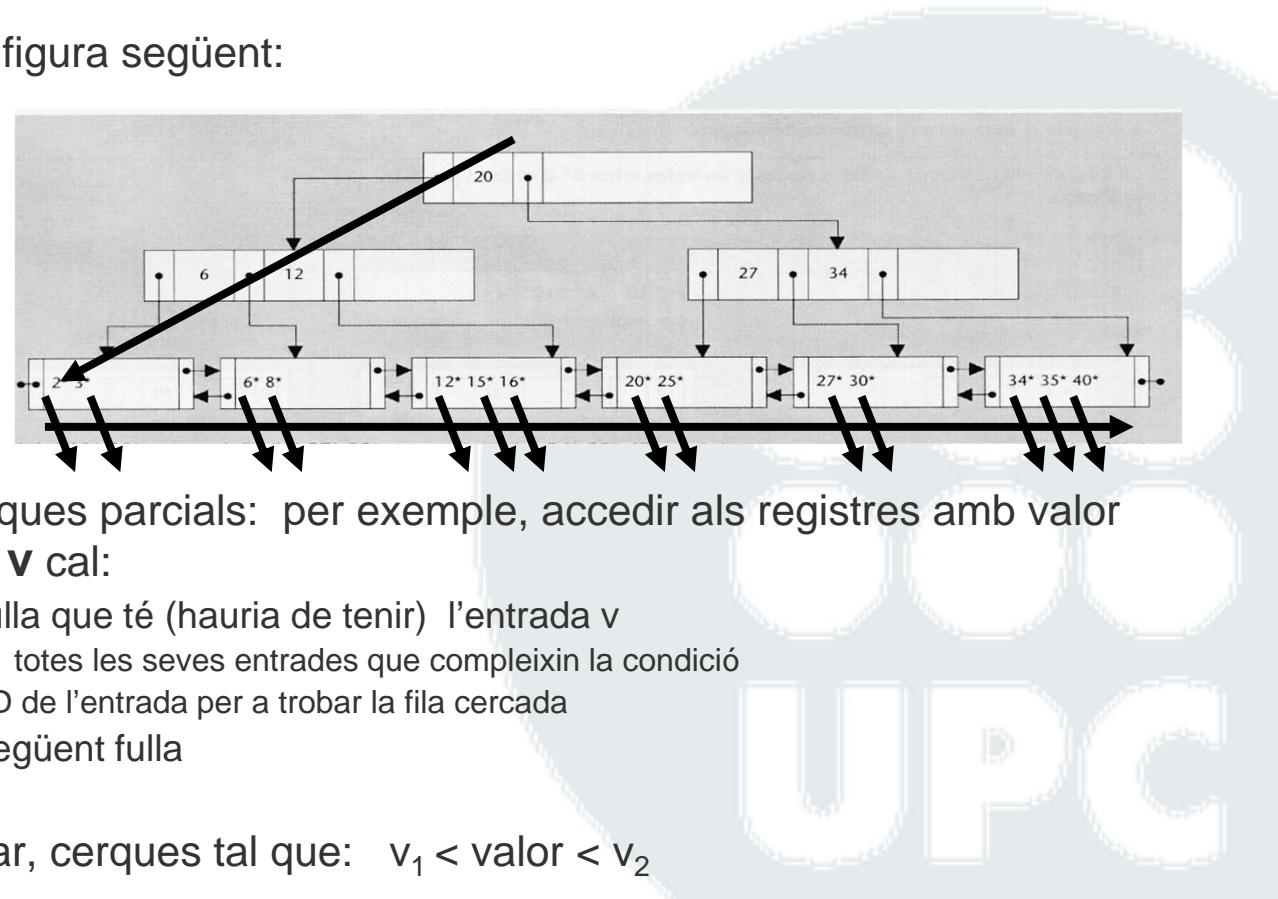
- Lectures: El nombre de pàgines a les que cal accedir és igual al nivell de la fulla on hauria d'estar + 1 accés a la pàgina de dades (en cas d'existir el valor).

## Arbres B+ Accés seqüencial per valor

- Per fer un accés seqüencial per valor total cal:
  - Localitzar la fulla de més a l'esquerra
    - Recuperar totes les seves entrades
    - Usar el RID de cada entrada per a trobar la fila cercada
  - Localitzar la fulla següent

En l'arbre de la figura següent:

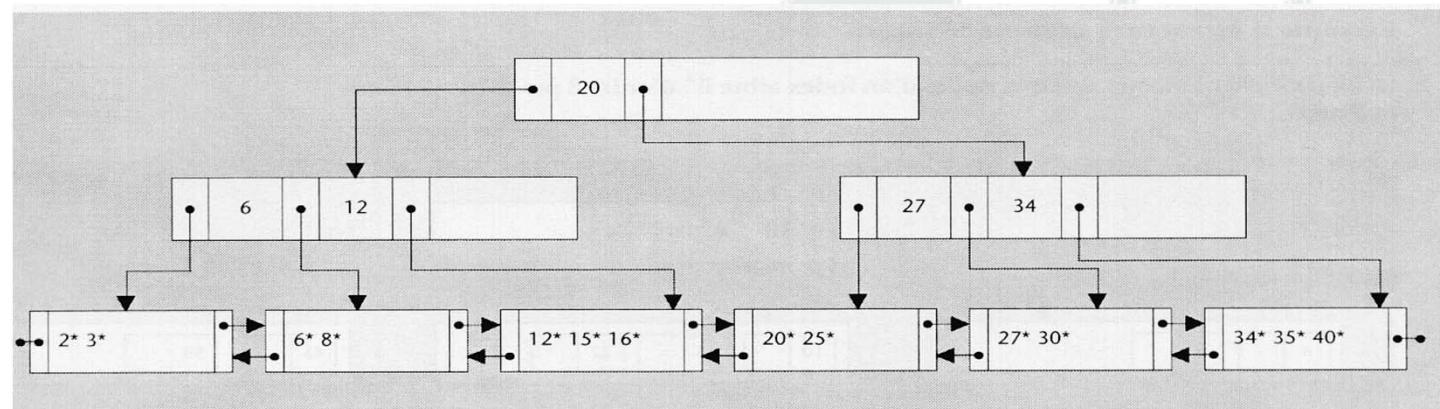
- 2, 3, 6, 8, .....



- Per tal de fer cerques parcials: per exemple, accedir als registres amb valor més gran o igual  $v$  cal:
  - Localitzar la fulla que té (hauria de tenir) l'entrada  $v$ 
    - Recuperar totes les seves entrades que compleixin la condició
    - Usar el RID de l'entrada per a trobar la fila cercada
  - Localitzar la següent fulla
- De manera similar, cerques tal que:  $v_1 < \text{valor} < v_2$

## Arbres B+ Millors de rendiment

- Per localitzar una entrada en una fulla d'un arbre, el nombre de nodes a recórrer és igual al nivell de la fulla!!!
  - Si reduïm el nivell de les fulles ...
- **Propietat 1:** Tots els nodes de l'arbre, excepte l'arrel (per què?), han d'estar plens com a mínim al 50%.
  - En un arbre d'ordre  $d \rightarrow$  almenys  $d$  valors per node
  - El següent arbre d'ordre 2 compleix aquesta propietat



- Que passaria en l'extrem si l'arbre fos binari?
- **Propietat 2:** Equilibrats. Totes les fulles al mateix nivell.
  - El nombre de nodes a recórrer és sempre el mateix

## Arbres B+ Emmagatzematge

- Espai d'índexs
- La mida dels nodes depèn de l'ordre i de la mida dels valors i apuntadors
  - Nodes molts grans → un arbre tingui pocs nivells
  - Nodes molts grans → potser més d'una E/S

La mida habitual d'un node coincideix amb la mida de les pàgines. **Cada node s'emmagatzema en una pàgina.** Per tant, en un arbre de **3 nivells, calen 3 E/S** per localitzar la fulla
- En cas que l'arrel estigui a memòria → 1 accés menys
- Suposant pàgines de 4Kb (4096b) → ordres **d** entre 50 i 100
  - Nodes interns:  $2d * (\text{mida valor}) + (2d + 1) * \text{mida apuntador fulla} \leq 4096b$   
»  $2d * (20) + (2d + 1) * 3 \leq 4096 ; d \leq 88$
  - Nodes fulla:  $2d * (\text{mida valor} + \text{mida RID}) + 2 * \text{mida apuntador fulla} \leq 4096b$   
»  $2d * (20 + 4) + 6 \leq 4096 ; d \leq 85$
  - Ordre òptim. Simplificació: Escollim  $d = 85$
  - Quants registres podem indexar i quin cost té localitzar-los???

## Arbres B+ Exemple

- Suposem que  $d = 50$  i que cada node té una ocupació 69%
  - Nivell 1: 1 node amb 69 valors i 70 apuntadors
  - Nivell 2: 70 nodes amb 69 valors i 70 apuntadors cadascun
  - Nivell 3: 4900 nodes amb 69 entrades

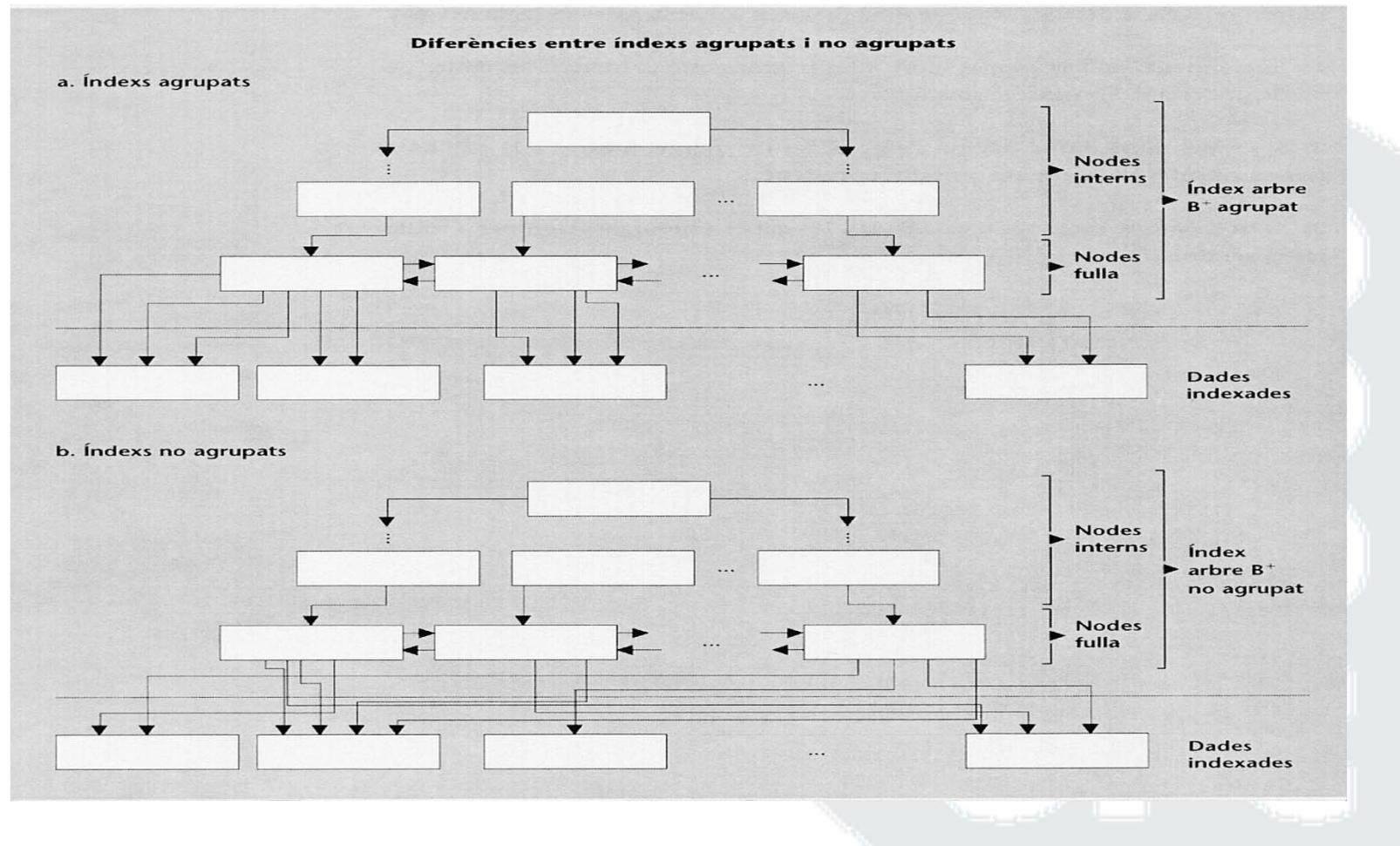
$$4900 * 69 = 338.100 \text{ entrades}$$

**Localitzar 1 registre entre 338100 es pot fer amb 3 accessos a pàgines de l'índex  
(+1 per accedir al registre –pàgina de dades)  
(-1 en cas que l'arrel estigui a memòria)**

- I amb 4 nivells i 69% ocupació quantes entrades?  
Nivell 4: 343.000 nodes amb 69 entrades = 23667000
- Arbre B+  
 $\approx \log_{70} 23667000 = \ln 23667000 / \ln 70 = 4;$   
 $\approx \log_{70} 338100 = 3$

## Índexs agrupats

- Un índex agrupat (*cluster*) és aquell en què les dades que indexa estan ordenades físicament segons l'accés seqüencial per valor que proporciona



## Índexs agrupats

- Quants índex agrupats podem tenir sobre una taula?
- Problema: Mantenir l'ordre físic?
  - Cost prohibitiu!
  - Solució pràctica:
    - Deixar % espai lliure a pàgines
    - Si s'omple, a pàgines d'excedents encadenades
    - Si % pàgines excedents elevat, regeneració



## Implementació dels accessos per diversos valors: Accessos directes, Estratègia intersecció de RIDs

Empleats(nemp, nom, despatx, sou)

Arbre B<sup>+</sup> sobre despatx

Arbre B<sup>+</sup> sobre sou

```
SELECT *
FROM Empleats
WHERE despatx = 150
AND sou = 200000
```

- **Estratègia Intersecció de RIDs:** Us de dos índexs arbres B+
  - Obtenir el conjunt dels RIDs tals que despatx = 150
  - Obtenir el conjunt dels RIDs tals que sou = 200000
  - Intersecció entre els dos conjunts de RIDs, obtenint els RIDs dels empleats que compleixen les dues condicions.

En general aquesta estratègia dóna un bon rendiment, però si hi ha molts RIDs que compleixen una condició (200000) i pocs les dues (150), és ineficient

## Implementació dels accessos per diversos valors: Accessos directes, Estratègia índex multi-atribut

Empleats(nemp, nom, despatx, sou)

Índex amb valors compostos  
pels atributs [num\_despatx, sou].  
Els valors de l'índex seran valors  
com ara [150, 200000], [150, 300000],  
[200, 100000], [200, 150000] ...

```
SELECT *  
FROM Empleats  
WHERE despatx = 150  
AND sou = 200000
```

### ■ Estratègia Índex (multiatribut)

Aquests índexs tenen la mateixa estructura **que els altres**, però **V** seran  
llistes de elements  $[v_1, v_2, \dots]$

La gestió de l'índex fa necessari establir una relació d'ordre lineal entre les  
llistes: s'ordenen primer segons el primer element, entre els que tenen  
el mateix valor pel primer, s'ordenen segons el segon element, ...

## Implementació dels accessos per diversos valors: Accessos seqüencials i mixtos

### ■ Estratègia Índex multi-atribut

Empleats(nemp, nom, despatx, sou)  
Índex multiatribut sobre [despatx, sou]

Els valors de l'índex seran valors  
com ara [150, 200000], [150, 300000],  
[200, 100000], [200, 150000] ...

```
SELECT *  
FROM Empleats  
ORDER BY despatx, sou
```

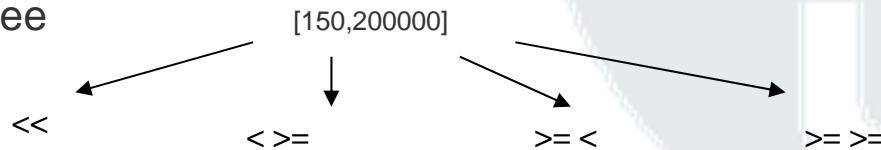
```
SELECT *  
FROM Empleats  
WHERE despatx = 150  
AND sou > 200000
```

```
SELECT *  
FROM Empleats  
ORDER BY sou, despatx
```

```
SELECT *  
FROM Empleats  
WHERE despatx > 150  
AND sou = 200000
```

### ■ Estratègia índex multi-atribut multi-dimesional: no ordre lineal!!

Ex: Quad-tree

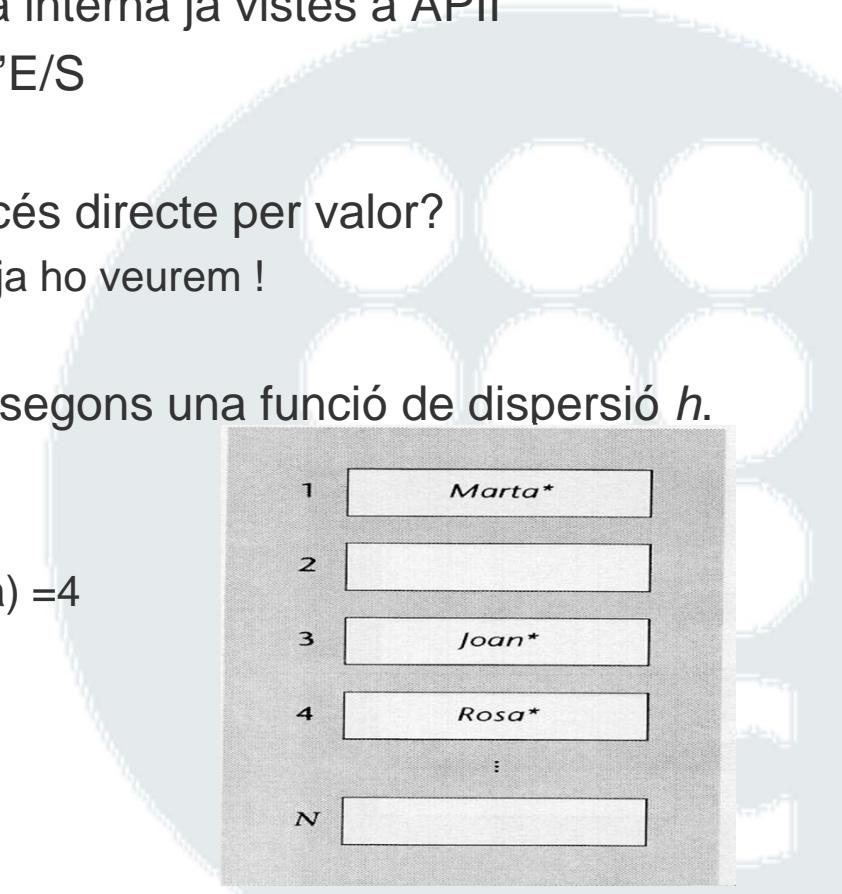


## Dispersió estàtica (Hash estàtic) Introducció

- Un altre tipus d'índexs per a facilitar l'accés directe per valor, però no accés seqüencial.
- Dispersió per a fer cerques a memòria interna ja vistes a APII  
BD: cerques al disc: Cal minimitzar l'E/S
- Per què un nou tipus i només per l'accés directe per valor?
  - Eficiència !  $\approx 1,1 + 1$  versus  $2 + 1$  ja ho veurem !
- Les entrades es col·loquen a pàgines segons una funció de dispersió  $h$ .

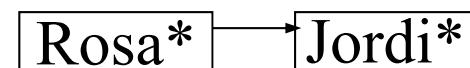
Número de pàgina  $p = h(v)$      $p: 1..N$

Suposem  $h(joan) = 3$ ;  $h(marta)=1$  i  $h(rosa) = 4$



## Dispersió Estàtica Estructura

- $h$ 
  - Qualsevol de les vistes a APII: típicament  $\text{mod } N + 1$
  - Si alfanumèric Suma ponderada de tots els caràcters
  
- Sinònims
  - $v_i \neq v_j \rightarrow h(v_i) = h(v_j)$ 
    - $h(\text{rosa})= 4$     $h(\text{jordi})=4$
    - APII: sinònims s'encadenen a la posició de la taula de dispersió.



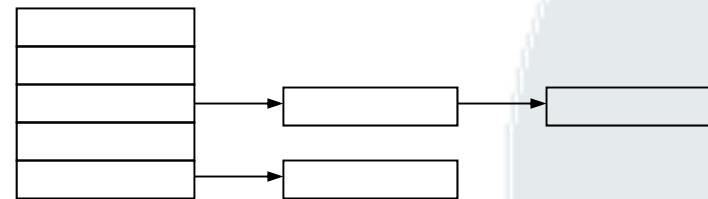
- Això significaria dos accessos a disc !
- BD: Les pàgines poden contenir diverses entrades.  
En particular, els sinònims.  $L$  és el número d'entrades per pàgina.

	1	2	...	$L$
1	Marta*			
2				
3	Joan*			
4	Rosa*	Jordi*		
:	:			
$N$				

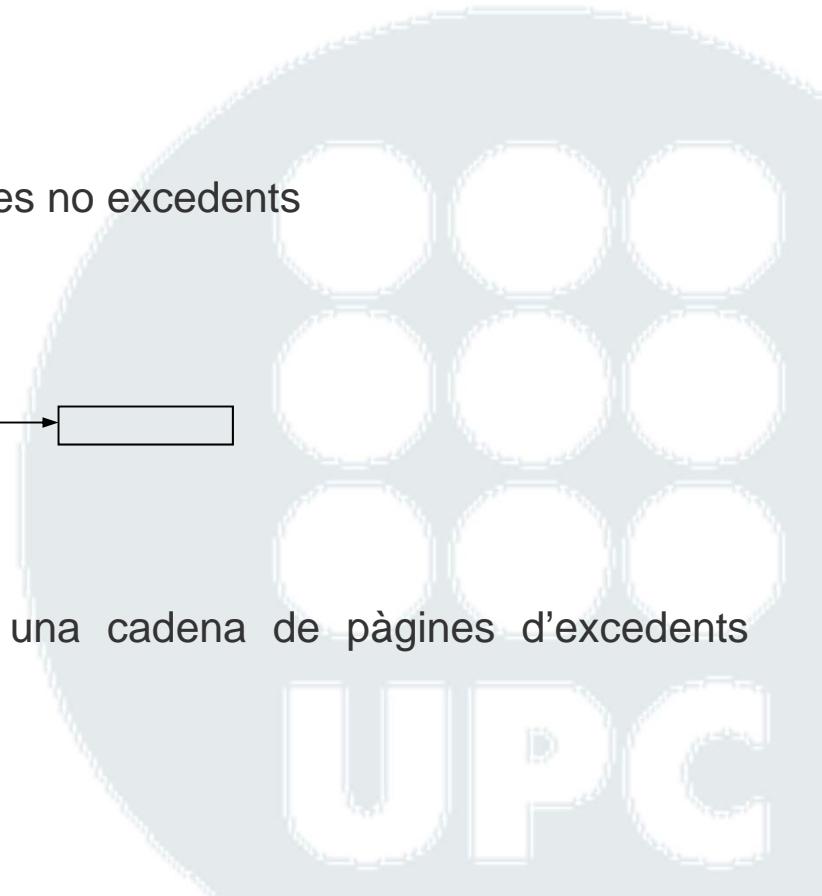


## Dispersió Estàtica      Estructura

- Excedent: Sinònims que no hi caben.
- Els excedents es col·loquen a altres pàgines segons una determinada política. Diverses: *open addressing*, ***separate chaining***, *coalesced chaining*...
- Dos tipus de pàgines:
  - Primàries: contindran totes les entrades no excedents
  - D'excedents: tots els excedents

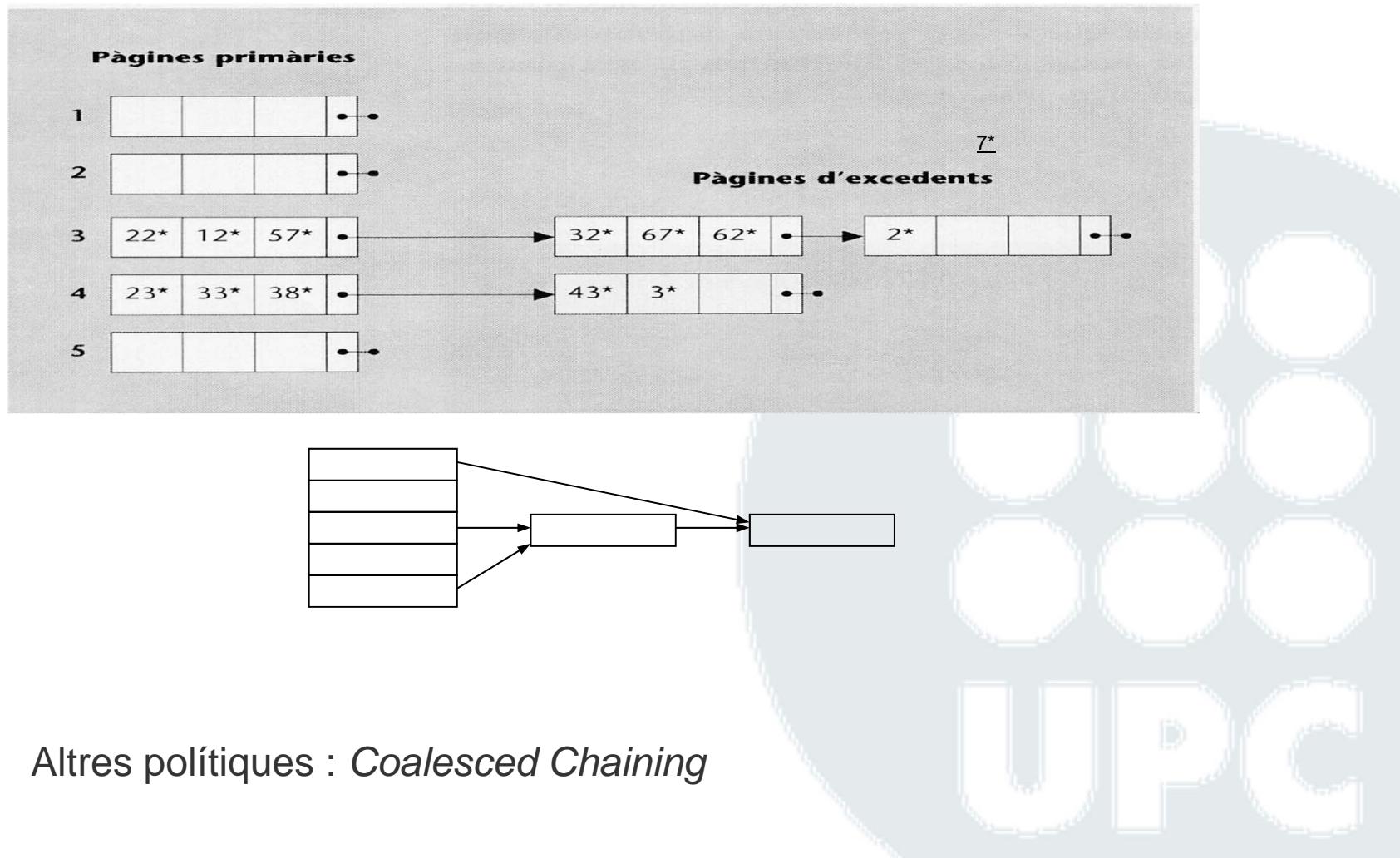


–Per a cada pàgina primària hi ha una cadena de pàgines d'excedents separades.



## Dispersió Estàtica Exemple

- N = 5; L = 3; h(v) = (v mod N) + 1; 22,23,12,57,33,38,32,67,62,2,43,3,7



## Emmagatzematge i Cost

- Cost de localitzar una entrada depèn de si és o no excedent !
  - Primària: 1 E/S
  - Excedent: més d'una
- Interessa pocs excedents. Cal una **L gran**, però limitat per la mida de la pàgina.
- Interessa pocs excedents: més espai del necessari a les pàgines primàries: **N gran !, és a dir**

Factor de càrrega  $C = M / (N \cdot L)$

- Numerador: M nombre d'entrades que s'esperen
- Denominador:  $N \cdot L$  nombre d'entrades que caben a les pàgines primàries

un **C petit, però**

Com més baix és el factor de càrrega,  
més espai es necessita

		Mitjana d'E/S per a entrades que són a l'índex										
		C	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,95
		L										
1		1,0500	1,1000	1,1500	1,2000	1,2500	1,3000	1,350	1,400	1,450	1,5	
2		1,0063	1,0242	1,0520	1,0883	1,1321	1,1823	1,238	1,299	1,364	1,4	
3		1,0010	1,0071	1,0216	1,0458	1,0806	1,1259	1,181	1,246	1,310	1,4	
4		1,0002	1,0023	1,0097	1,0257	1,0527	1,0922	1,145	1,211	1,290	1,3	
5		1,0000	1,0008	1,0046	1,0151	1,0358	1,0699	1,119	1,186	1,286	1,3	
10		1,0000	1,0000	1,0002	1,0015	1,0070	1,0226	1,056	1,115	1,206	1,3	
20		1,0000	1,0000	1,0000	1,0000	1,0005	1,0038	1,018	1,059	1,150	1,2	
50		1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,001	1,015	1,083	1,2	

D. Knuth (1973). "Sorting and Searching". *The Art of Computer Programming* (vol. 3). Reading(Massachusetts): Addison-Wesley.

## Dispersió Estàtica $M \rightarrow C, L, N$

- $M = 338.000$  entrades;

Mida valor= 37 bytes; RID = 4 bytes; apuntadors=3 bytes;

- Càcul de L

mida pàgina = mida entrades \* L + mida apuntador excedent

$$2048 = (37 + 4) * L + 3 \rightarrow L = 49$$

- Determinar C

$$C = 0.6 \quad 1,000$$

$$C = 0.8 \quad 1,015$$

$$C = 0.9 \quad 1,083$$

- Calcular N

$$N = M / C * L = 338.000 / 0.9 * 49 = 7665 \text{ pàgines primàries}$$

$$N = M / C * L = 338.000 / 0.8 * 49 = 8623 \text{ pàgines primàries}$$

$$N = M / C * L = 338.000 / 0.6 * 49 = 11487 \text{ pàgines primàries}$$

- N convé que sigui primer ( o al menys no tingui divisors entre els primers més petits de 20).

$$P = h(v) = (v \bmod N) + 1$$

$$v = N * q + P$$

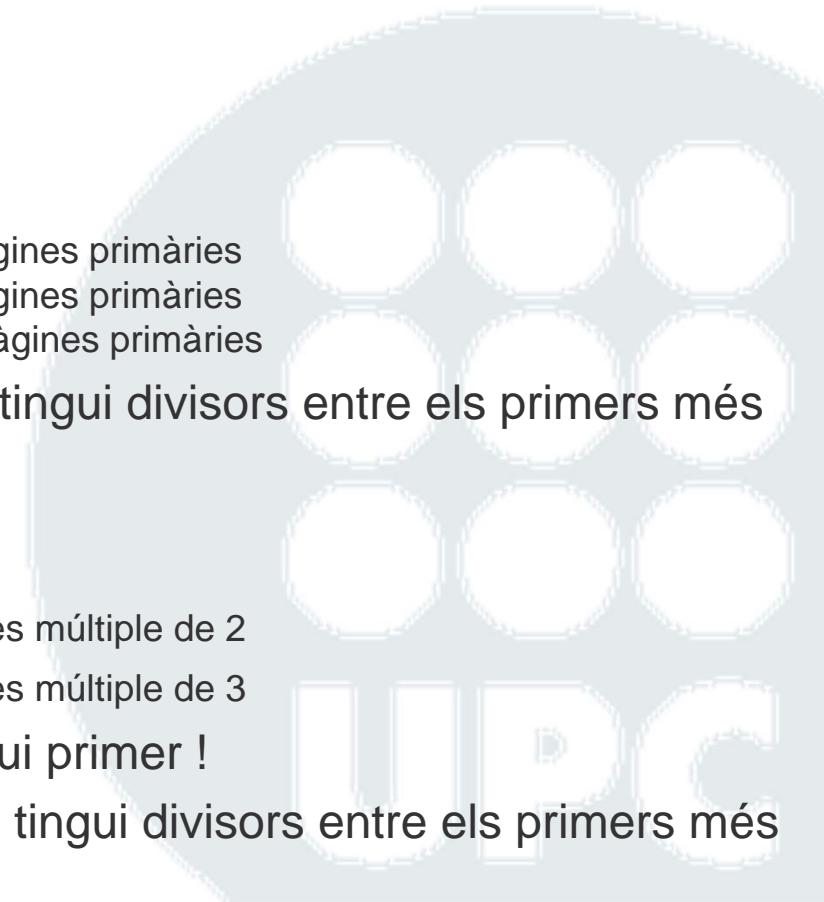
$$P = v - N * q$$

Si v és múltiple de 2 i N és múltiple de 2, p és múltiple de 2

Si v és múltiple de 3 i N és múltiple de 3, p és múltiple de 3

N el número més gran de 7665 que sigui primer !

N el número més gran de 7665 que no tingui divisors entre els primers més petits de 20!



Per simplificar, considerarem que tots els sinònims caben en una pagina

## Cost accés directe per valor: Índex Hash, no valors repetits

- Índex Hash sobre un atribut  $b$  d'una taula  $R$ , l'atribut  $b$  no pren valors repetits
- Cost = Cost accés índex + Cost accés fitxer de dades = 1 + 1

## Cost accés directe per valor: Índex Hash, valors repetits

- Índex Hash sobre un atribut  $b$  d'una taula  $R$ , l'atribut  $b$  pren valors repetits.
- Màxim  $L$  repetits
- Cost = Cost accés índex + Cost accés fitxer de dades =  $1 + |R(b=Y)|$

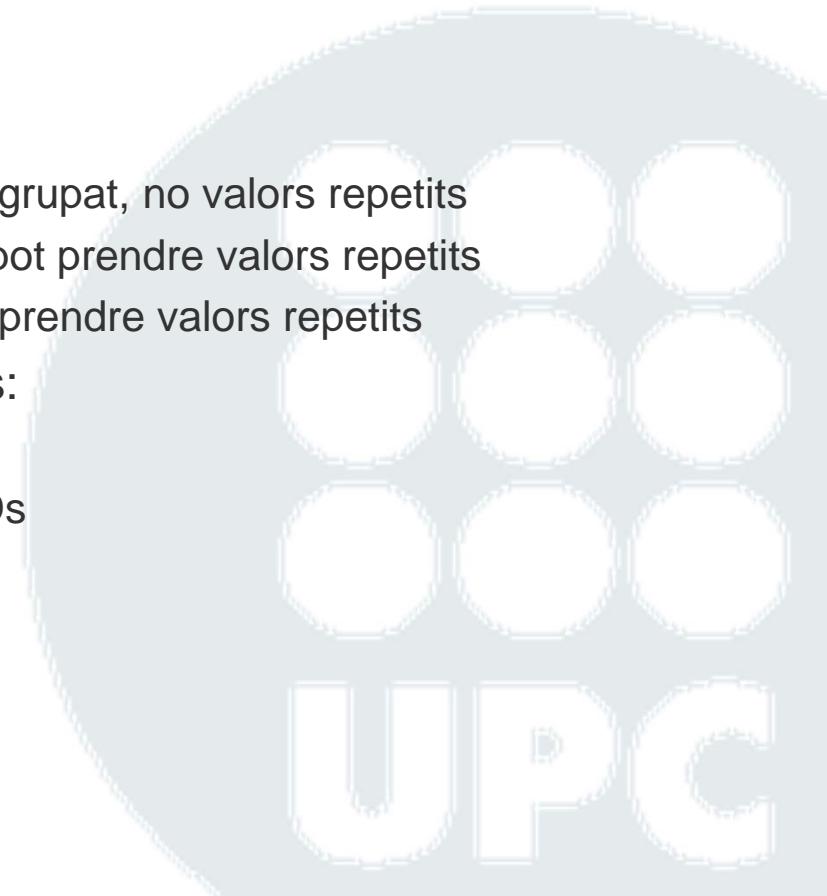


## Els índexs en un SGBD concret

- Per implementar accessos per valor, directes o seqüencials, la majoria d'SGBDs proporcionen índexs arbre B+ per un atribut que poden ser agrupats o no
  - Creació d'un índex:  
Empleats(nemp, nom, despatx, sou)  
CREATE INDEX index\_despatx ON Empleats(despatx)
  - Per a que l'índex sigui hash:  
CREATE INDEX index\_despatx ON Empleats USING HASH (despatx)
  - Creació d'un índex agrupat:  
CREATE INDEX CLUSTER index\_despatx ON Empleats(despatx)
  - Creació d'un índex que no admeti valors repetits:  
CREATE UNIQUE INDEX index\_despatx ON Empleats(despatx)
  - Creació d'un índex multiatribut:  
CREATE INDEX index\_despatx\_sou ON Empleats(despatx,sou)

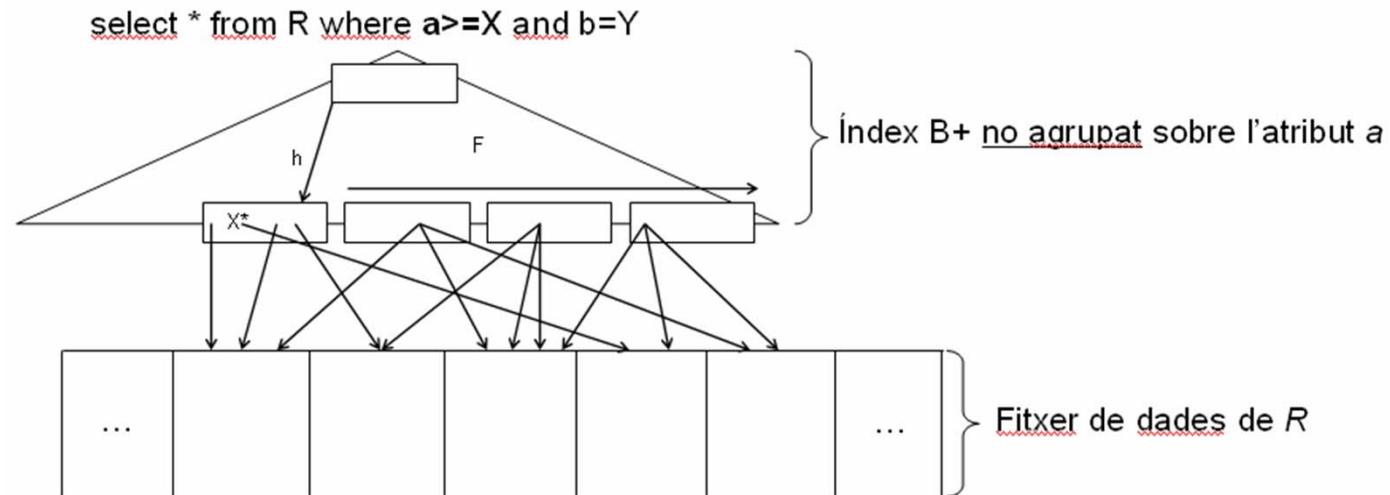
## Annex

- Cost accés seqüencial per valor:
  - Índex arbre B+, no agrupat
  - Índex arbre B+, agrupat
- Cost accés directe per valor:
  - Índex arbre B+, agrupat/no agrupat, no valors repetits
  - Índex arbre B+, no agrupat, pot prendre valors repetits
  - Índex arbre B+, agrupat, pot prendre valors repetits
- Cost accés per diversos valors:
  - Alternatives possibles
  - Estratègia intersecció de RIDs



## Cost accés seqüencial per valor: Índex arbre B+, no agrupat

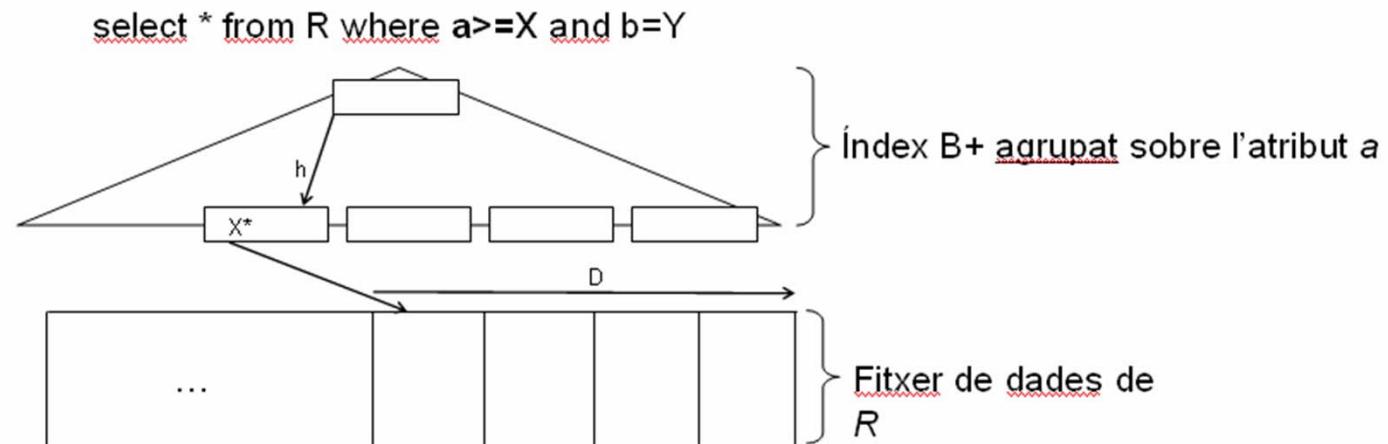
- Índex B+ no agrupat sobre un atribut  $a$  d'una taula  $R$ , l'atribut  $a$  pot prendre (o no) valors repetits



- Cost = Cost accés índex + Cost accés fitxer de dades =  $(h + F) + |R(a>=X)|$
- $h$  alçada índex B+,  $F$  nombre de nodes fulla addicionals (a més del primer node fulla) de l'índex B+ que cal recórrer
- $|R(a>=X)|$  nombre de files de la taula  $R$  que verifiquen la condició  $a>=X$  expressada a la consulta
- El cost d'accés a l'índex es pot disminuir en 1 unitat si l'arrel està a memòria

## Cost accés seqüencial per valor: Índex arbre B+, agrupat

- Índex B+ agrupat sobre un atribut  $a$  d'una taula  $R$ , l'atribut  $a$  pot prendre (o no) valors repetits

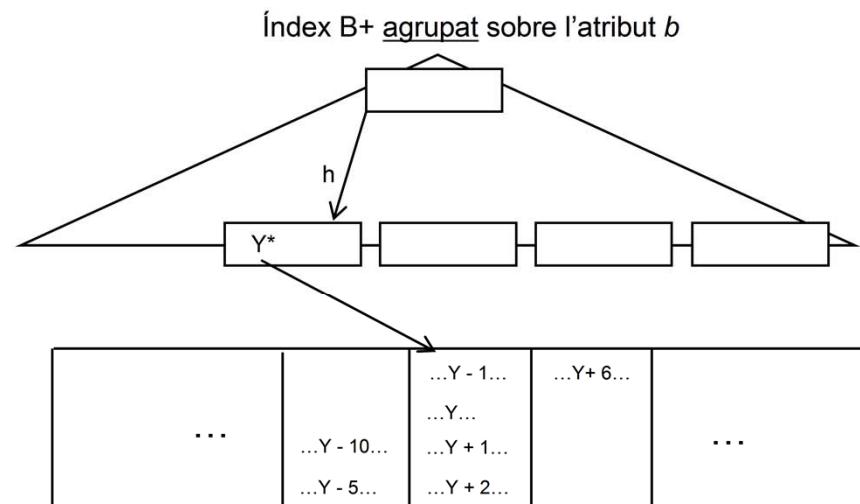


- Cost = Cost accés índex + Cost accés fitxer de dades =  $h + D$
- $h$  alçada índex B+,  $D$  nombre de pàgines del fitxer de dades que cal recórrer
- $D = \lceil |R(a \geq X)|/f \rceil$
- $|R(a \geq X)|$  nombre de files de la taula  $R$  que verifiquen la condició  $a \geq X$  expressada a la consulta
- $f$  factor de bloqueig del fitxer de dades (nombre de registres per pàgina)
- El cost d'accés a l'índex es pot disminuir en 1 unitat si l'arrel està a memòria

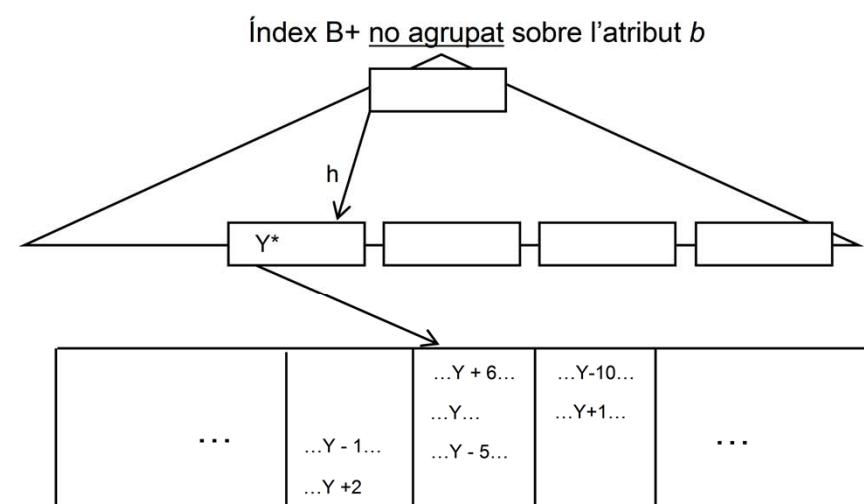
## Cost accés directe per valor: Índex arbre B+, no valors repetits

- Índex B+ agrupat/no agrupat sobre un atribut  $b$  d'una taula  $R$ , l'atribut  $b$  no pren valors repetits
- Cost = Cost accés índex + Cost accés fitxer de dades =  $h + 1$
- $h$  alçada de l'arbre B+
- El cost d'accés a l'índex es pot disminuir en 1 unitat si l'arrel està a memòria

`select * from R where a>=X and b=Y`



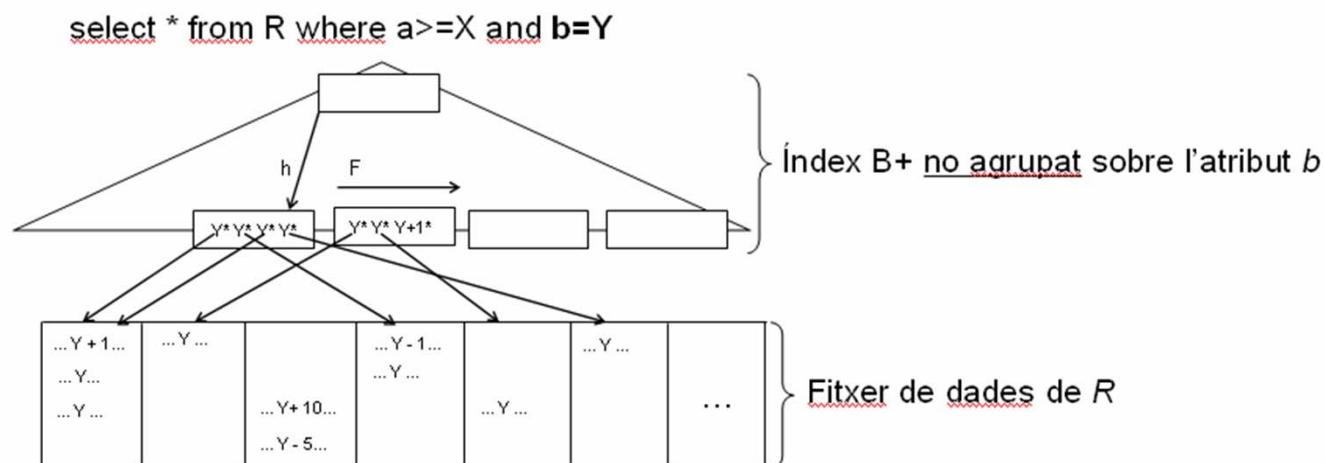
Fitxer de dades de  $R$   
(les dades estan ordenades segons el valor de l'atribut  $b$ )



Fitxer de dades de  $R$   
(les dades no estan ordenades segons el valor de l'atribut  $b$ )

## Cost accés directe per valor: Índex arbre B+, no agrupat, pot prendre valors repetits

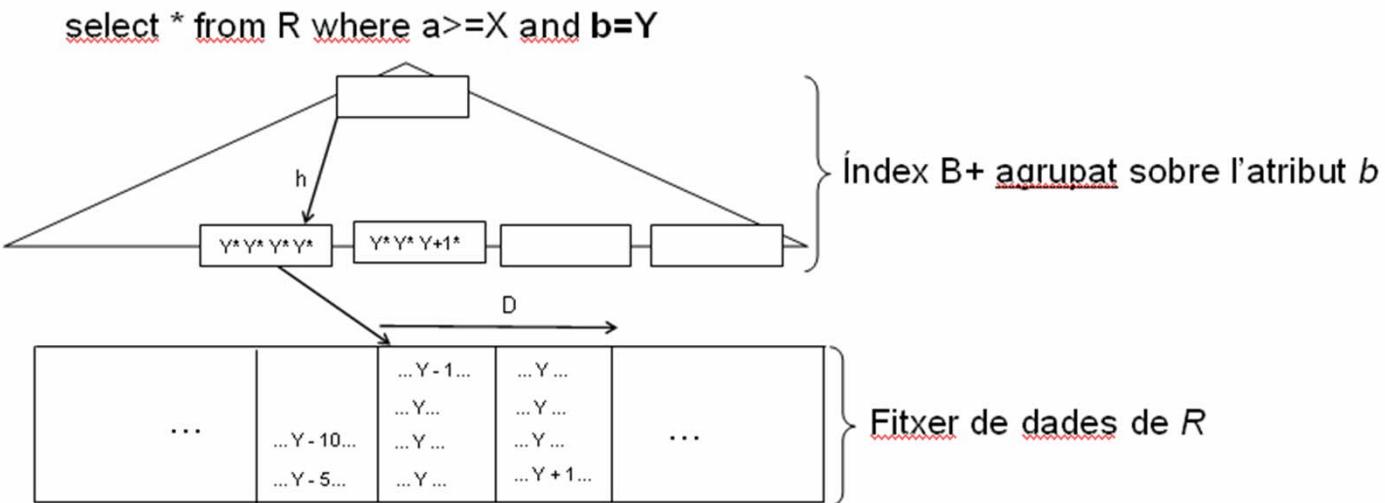
- Índex B+ no agrupat sobre un atribut  $b$  d'una taula  $R$ , l'atribut  $b$  pot prendre valors repetits



- Cost= Cost accés índex + Cost accés fitxer de dades=  $(h + F) + |R(b=Y)|$
- $h$  alçada índex B+,  $F$  nombre de nodes fulla addicionals (a més del primer node fulla) de l'índex B+ que cal recórrer
- $|R(b=Y)|$  nombre de files de la taula  $R$  que verifiquen la condició  $b=Y$  expressada a la consulta
- El cost d'accés a l'índex es pot disminuir en 1 unitat si l'arrel està a memòria

## Cost accés directe per valor: Índex arbre B+, agrupat, pot prendre valors repetits

- Índex B+ agrupat sobre un atribut  $b$  d'una taula  $R$ , l'atribut  $b$  pot prendre valors repetits



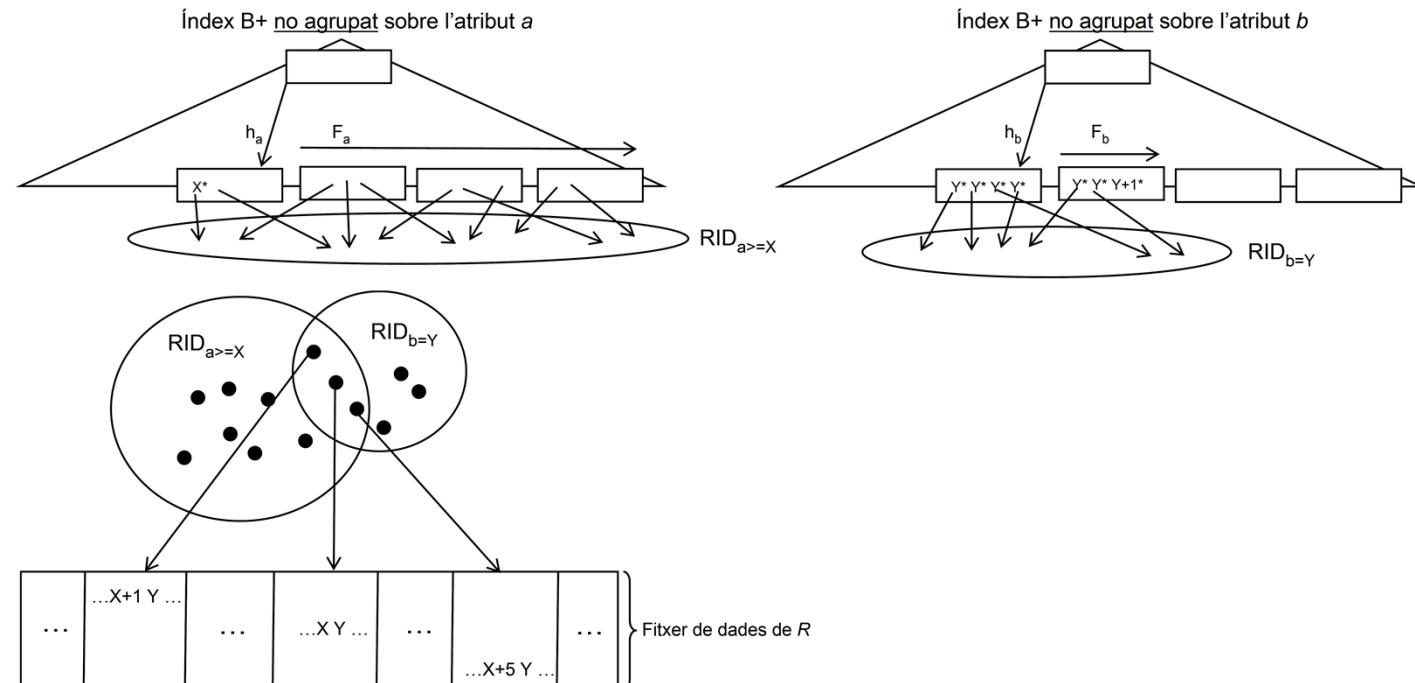
- Cost = Cost accés índex + Cost accés fitxer de dades =  $h + D$
- $h$  alçada índex B+,  $D$  nombre de pàgines del fitxer de dades que cal recórrer
- $D = \lceil |R(b=Y)|/f \rceil$
- $|R(b=Y)|$  nombre de files de la taula  $R$  que verifiquen la condició  $b=Y$  expressada a la consulta
- $f$  factor de bloqueig del fitxer de dades (nombre de registres per pàgina)
- El cost d'accés a l'índex es pot disminuir en 1 unitat si l'arrel està a memòria

## Cost accés per diversos valors: Alternatives possibles

- Imaginem que tenim la consulta següent:  
`select * from R where a>=X and b=Y`
- Disposem de dos índexs B+ no agrupats, un sobre *a* i un altre sobre *b*, *b* pot prendre valors repetits
- Per resoldre la consulta, usant els índexs, tenim diverses alternatives:
  - Utilitzar només l'índex B+ definit sobre *a*
  - Utilitzar només l'índex B+ definit sobre *b*
  - Utilitzar simultàniament tots dos índexs B+:
    - Accés índex B+ sobre *a*: recuperem els RID de les files del fitxer de dades que verifiquen la condició  $a \geq X$ . Anomenem  $RID_{a \geq X}$  el conjunt d'RID trobats
    - Accés índex B+ sobre *b*: recuperem els RID de les files del fitxer de dades que verifiquen la condició  $b=Y$ . Anomenem  $RID_{b=Y}$  el conjunt d'RID trobats
    - Intersecció dels dos conjunts de RID:  $RID_{a \geq X} \cap RID_{b=Y}$ , obtenim els RID de les files del fitxer de dades que simultàniament verifiquen les dues condicions ( $a \geq X$  and  $b=Y$ )
    - Es desencadenen els accessos corresponents ( $RID_{a \geq X} \cap RID_{b=Y}$ ) al fitxer de dades de *R*
- No es pot saber *a priori* quina és la millor alternativa, pot variar en funció de la consulta concreta a resoldre

## Cost accés per diversos valors: Estratègia intersecció de RIDs

- Cost = Cost accés índex<sub>1</sub>+ ... + Cost accés índex<sub>n</sub> + Cost accés fitxer de dades
- Sobre el nostre exemple: select \* from R where **a>=X** and **b=Y**



- Cost =  $(h_a+F_a) + (h_b+F_b) + |RID_{a>=X} \cap RID_{b=Y}|$
- $|RID_{a>=X} \cap RID_{b=Y}|$  és equivalent a  $|R(a>=X \wedge b=Y)|$  que és el nombre de files de la taula  $R$  que verifiquen la condició expressada a la consulta ( $a>=X$  and  $b=Y$ )
- El cost d'accés a cada índex es pot disminuir en 1 unitat si les arrels estan a memòria