

## Estructura de dades: Claus d'una relació

- **Superclau d'una relació:** Subconjunt dels atributs de l'esquema de la relació que identifica les tuples de l'extensió de la relació
- **Clau d'una relació:** Superclau de la relació que no té cap subconjunt propi que sigui també superclau  
També s'anomena **clau candidata** de la relació
- Exemples: EMPLEAT(DNI, NSS, Nom, Telèfon, Sou)  
Superclaus: {DNI, NSS, Nom, Telèfon, Sou}  
{DNI, Nom}, {DNI}, etc.  
Claus candidates: {DNI}, {NSS}  
DESPATX(Edifici, Número, Superfície)  

C6	119	15
C6	120	15
D4	119	10

  
Clau candidata: {Edifici, Número}

## Estructura de dades: Claus d'una relació - Claus foranes

- Les tuples de les relacions d'una base de dades poden requerir **connexions** entre elles
- Exemple: EMPLEAT i DEPARTAMENT  
cada empleat està assignat a un departament  
un empleat pot tenir un altre empleat que li fa de "cap"  

The diagram illustrates a foreign key relationship. At the top, a box labeled 'DEPARTAMENT(NomDep, ...)' contains the attributes 'Producció' and 'Vendes'. Below it, a box labeled 'EMPLEAT(DNI, ...)' contains the attributes '40.444.255', '33.567.711', '55.898.425', and '77.232.144'. A curved arrow points from the 'NomDep' attribute in the DEPARTAMENT box to the '40.444.255' attribute in the EMPLEAT box. Another curved arrow points from the 'NomDep' attribute in the DEPARTAMENT box to the '33.567.711' attribute in the EMPLEAT box. A third curved arrow points from the 'NomDep' attribute in the DEPARTAMENT box to the '55.898.425' attribute in the EMPLEAT box. A fourth curved arrow points from the 'NomDep' attribute in the DEPARTAMENT box to the '77.232.144' attribute in the EMPLEAT box. To the right of the EMPLEAT table, there is a separate row with attributes 'EmpleatCap, DepAssig' and values 'NULL', '40.444.255', '33.567.711', and 'Vendes'.
- Una **clau forana d'una relació** és un subconjunt dels atributs de l'esquema de la relació que **referencia** una clau primària d'una altra relació o de la pròpia relació

## Estructura de dades: Claus d'una relació - Clau primària i alternatives

- Una de les claus candidates es designa **clau primària**
- **Clau alternativa:** Clau candidata no designada primaria
- Exemple:

EMPLEAT(DNI, NSS, Nom, Telèfon, Sou)

Claus candidates: {DNI}, {NSS}

Clau primària: {DNI}

Clau alternativa: {NSS}

- Convenció: Se subratllen els atributs que formen la clau primària

## Estructura de dades: Claus d'una relació - Claus foranes

- Una **clau forana** ha de complir que:
  - Té el mateix nombre d'atributs que la clau primària referenciada
  - Els atributs que la formen han de tenir dominis compatibles amb els de la clau primària referenciada

- Exemple:

DESPATX(Edifici, Número, Superfície)

EMPLEAT(DNI, ... ,  
, EdDespatx, NumDespatx)

## Regles d'integritat

- Regla d'integritat d'**entitat**
- Regla d'integritat **referencial**
- Regla d'integritat de **domini**

### Regles d'integritat: Regla d'integritat d'entitat

- Fa referència a les **claus primàries** de les relacions
- Estableix que:
  - Els atributs que formen part de la clau primària han de tenir **valors únics** en conjunt (no repetits)
  - Cap atribut de la clau primària pot prendre el **valor nul**
- Exemple:

DESPATX(Edifici, Número, Superfície)		
C6	119	15
C6	120	15
D4	119	10
<u>C6</u>	<u>119</u>	<u>12</u>
<u>NULL</u>	<u>NULL</u>	<u>25</u>
<u>C6</u>	<u>NULL</u>	<u>10</u>
<u>NULL</u>	<u>119</u>	<u>8</u>

- **Motivació:** la clau primària ha de servir per identificar les tuples d'una relació

### Regles d'integritat: Integritat referencial - restricció

- **No es permet** esborrar o modificar una clau primària referenciada en alguna clau forana
- Exemple:

CLIENT(NumClient, ...)	
10	
15	
18	

FACTURES_PENDENTS(NumFactura, ..., NumClient)	
1234	10
1235	10
1236	15

### Regles d'integritat: Regla d'integritat referencial

- Fa referència a les **claus foranes** de les relacions
- Estableix que:
  - Els valors d'una clau forana poden ser només valors de la **clau primària referenciada** o **valors nuls**
- Exemple:

DEPARTAMENT(NomDep, ...)	
Producció	Vendes
40.444.255	
33.567.711	
55.898.425	
77.232.144	
25.250.333	Marketing

EMPLEAT(DNI, ...)	
DepAssig	
40.444.255	Producció
33.567.711	Producció
55.898.425	Vendes
77.232.144	NULL
25.250.333	Marketing

- **Motivació:** les claus foranes han de servir per establir connexions entre les tuples

### Regles d'integritat: Manteniment de la integritat referencial

- Una **operació** d'inserció, esborrat o modificació pot conduir a un estat que no satisfaci les regles d'integritat
- Manteniment de la integritat:
  - **rebutjar** l'operació
  - acceptar-la i realitzar **accions compensatòries**
- En els casos següents:
  - **esborrat** d'una clau primària referenciada
  - **modificació** d'una clau primària referenciadapoden aplicar-se accions compensatòries per al manteniment de la **integritat referencial**
- Per a cada clau forana el dissenyador de la BD escollirà la política adequada en cas d'esborrat i en cas de modificació
- Algunes polítiques possibles:
  - **Restricció**
  - **Cascada**
  - **Anul·lació**

## Regles d'integritat: Integritat referencial - cascada

- En cas d'esborrar o modificar una clau primària referenciada en alguna clau forana, **s'esborren o modifiquen** totes les referencies
- Exemple:

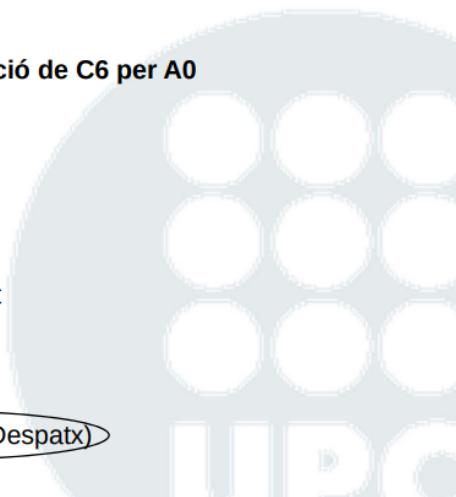
EDIFICI(NomEdif, Direcció)  
A0 ← C6  
D4

DESPATX(Edifici, Número, Superfície)  
A0 ← C6 119 15  
A0 ← C6 120 15  
D4 119 10

modificació de C6 per A0

- Aquesta política es pot aplicar recursivament

DESPATX(Edifici, Número, Superfície)  
EMPLEAT(DNI, ... , EdDespatx, NumDespatx)



## Regles d'integritat: Integritat referencial - cascada

- En cas d'esborrar o modificar una clau primària referenciada en alguna clau forana, **s'esborren o modifiquen** totes les referencies
- Exemple:

EDIFICI(NomEdif, Direcció)

esborrat de C6

C6  
D4

DESPATX(Edifici, Número, Superfície)

119 15

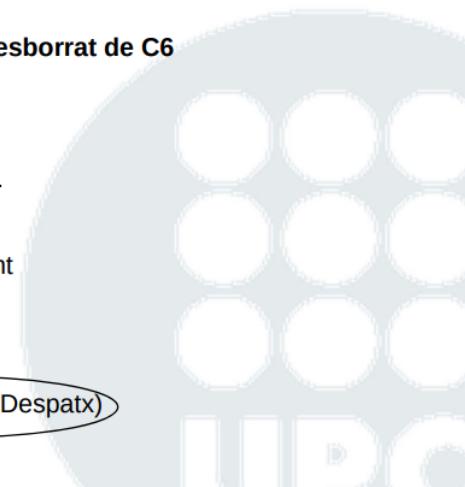
120 15

119 10

- Aquesta política es pot aplicar recursivament

DESPATX(Edifici, Número, Superfície)

EMPLEAT(DNI, ... , EdDespatx, NumDespatx)



## Regles d'integritat: Integritat referencial - anul.lació

- En cas d'esborrar o modificar una clau primària referenciada en alguna clau forana, s'assignen **valors nuls** a totes les referències

- Exemple:

VENEDOR(NumVenedor, ...)

1  
2  
3

esborrat del venedor 1

CLIENT(NumClient, ... , NumVenedor)

23  
35  
38  
42  
50

1  
2  
3

NULL  
NULL

## Base de dades exemple

- Clau primària:** Cada taula té una clau primària que permet identificar les files de la taula. Per ex. **num\_dpt** és la clau primària de la taula departaments. Això vol dir que cada departament té un num\_dpt que ha de ser únic entre tots els departaments, és a dir mai hi haurà dos departaments amb el mateix número de departament.
- Clau forana.** Una clau forana permet relacionar les files de dues taules. Per ex. **num\_dpt** és una clau forana de la taula empleats que referencia la taula departaments. Això vol dir que entre les dades d'un empleat hi haurà també el departament al que pertany, i això ens permetrà saber el departament on treballa un empleat, i també els empleats que treballen a un departament.

departaments(num\_dpt, nom\_dpt, planta, edifici, ciutat\_dpt)

1	DIRECCIO	10	PAU CLARIS	BARCELONA
2	DIRECCIO	8	RIOS ROSAS	MADRID
3	MARQUETING	1	PAU CLARIS	BARCELONA

projectes(num\_proj, nom\_proj, producte, pressupost)

1	IBDTEL	TELEVISIO	1000000
2	IBDVID	VIDEO	500000

empleats(num\_empl, nom\_empl, sou, ciutat\_empl, num\_dpt, num\_proj)

1	CARME	400000	MATARO	1	1
2	EUGENIA	350000	TOLEDO	2	2
3	JOSEP	250000	SITGES	3	1

## Creació d'una taula

**CREATE TABLE <nom\_taula>**

```
(<nom_columna> <tipus_dades> [<restriccions_col>] [<val_per_defecte>]  
[, <nom_columna> <tipus_dades> [<restriccions_col>] [<val_per_defecte>]...]  
[<restriccions_taula>]);
```

- **tipus\_dades:** INTEGER, FLOAT(precisió), REAL, CHAR(n), NUMERIC(precisió, escala), DECIMAL(precisió, escala), SMALLINT, DOUBLE PRECISION, VARCHAR(n), DATE,....
- **val\_per\_defecte:** Valor per defecte d'una columna per a una fila que s'insereix a la taula.

DEFAULT { <literal> | NULL }.

## Creació d'una taula: Restriccions de taula

- **restriccions\_taula:**

UNIQUE (<cols>)	El conjunt de les columnes especificades han de tenir valors únics entre les files de la taula
PRIMARY KEY (<cols>)	El conjunt de les columnes especificades formen la clau primària
FOREIGN KEY (<cols>) REFERENCES <taula> [<cols>]	El conjunt de columnes especificades formen una clau forana que referencia la taula indicada.
CHECK (<condicions>)	La taula ha de complir les condicions especificades. La condició pot referir-se a una o més columnes de la taula.
NOT NULL	La columna no pot tenir valors nuls

Les **restriccions de taula** poden referir-se a una o més columnes de la taula.

Així, en cas de restriccions que tenen a veure amb més d'una columna cal usar una restricció de taula.

Per exemple, en cas de claus primàries compostes per més d'un columna o condició (CHECK) que tenen a veure amb més d'una columna.

## Creació d'una taula: Exemple

**CREATE TABLE empleats**

```
( num_empl      INTEGER,  
  nom_empl     CHAR(30) NOT NULL,  
  sou          INTEGER DEFAULT 100000  
                CHECK (sou>80000),  
  ciutat_empl   CHAR(30),  
  num_dpt       INTEGER,  
  num_proj      INTEGER,  
  PRIMARY KEY (num_empl),  
  FOREIGN KEY (num_dpt) REFERENCES departaments(num_dpt),  
  FOREIGN KEY (num_proj) REFERENCES projectes(num_proj));
```

## Esborrat de files d'una taula

**DELETE FROM <taula>**

**WHERE <condicions>;**

- S'eliminen de la **taula** les files que compleixen les **condicions** especificades a la clausula **WHERE**.

## Inserció de files en una taula: Exemples

**INSERT INTO empleats**

**VALUES (4, 'RICARDO', 400000, 'BARCELONA', 1, 1);**

**INSERT INTO empleats (num\_empl, num\_dpt, num\_proj,  
 nom\_empl)**

**VALUES (11, 3, 2, 'NURIA');**

empleats	num_empl	nom_empl	sou	ciutat_empl	num_dpt	num_proj
	1	CARME	400000	MATARÓ	1	1
	2	EUGENIA	350000	TOLEDO	2	2
	3	JOSEP	250000	SITGES	3	1
	4	RICARDO	400000	BARCELONA	1	1
	11	NURIA	100000	NULL	3	2

Files inserides per la primera sentència

Files inserides per la segona sentència

## Modificació de files d'una taula: Exemples

```
UPDATE empleats  
SET sou = sou +10000  
WHERE num_dpt = 1;
```

```
UPDATE empleats  
SET sou = sou + 50000, ciutat_empl = 'VIC'  
WHERE num_empl = 11;
```

empleats(num_empl, nom_empl, sou, ciutat_empl, num_dpt, num_proj)					
1	CARME	410000	MATARO	1	1
2	EUGENIA	350000	TOLEDO	2	2
3	JOSEP	250000	SITGES	3	1
4	RICARDO	410000	BARCELONA	1	1
11	NURIA	150000	VIC	3	2

■ Files modificades per la primera sentència  
□ Files modificades per la segona sentència

## Consultes sobre una taula: Format bàsic

```
SELECT <columnes_a_seleccionar> | *  
FROM <taula_a_consultar>  
[ WHERE <condicions> ] ;
```

- El resultat de la consulta és el valor de les **columnes\_a\_seleccionar** de la **taula\_a\_consultar** únicament per a la fila o files que compleixen les **condicions** especificades a la clausula **WHERE**.
- En el cas de no posar la clausula **WHERE**, el resultat és el valor de les **columnes\_a\_seleccionar** per totes les files de la **taula\_a\_consultar**.
- Si posem un \* en lloc de **columnes\_a\_seleccionar** indica que estem interessats en totes les columnes de la **taula\_a\_consultar**.

## Operadors en les condicions

### ■ operadors

- aritmètics: \*, +, -, /
- de comparació: =, <, >, <=, >=, <>
- lògics: NOT, AND, OR
- altres:
  - <columna> BETWEEN <límit<sub>1</sub>> AND <límit<sub>2</sub>>
  - <columna> IN (<valor<sub>1</sub>>,<valor<sub>2</sub>> [....,<valor<sub>N</sub>>])
  - <columna> LIKE <característica>
  - <columna> IS [NOT] NULL

Aquests operadors poden sortir a les **condicions**

- En la clausula **WHERE** de les sentències d'esborrat (**DELETE**), modificació (**UPDATE**) i consulta (**SELECT**)
- En la clausula **CHECK** de les sentències de creació d'una taula (**CREATE TABLE**).

## Operadors en les condicions: Exemple

```
SELECT num_empl, nom_empl  
FROM empleats  
WHERE NOT(num_dpt = 2) AND  
( ciutat_empl IN ('MATARO', 'SITGES', 'BARCELONA') OR  
ciutat_empl LIKE 'V%') AND  
num_proj IS NOT NULL AND  
sou BETWEEN 400000 AND 500000;
```

empleats(num_empl, nom_empl, sou, ciutat_empl, num_dpt, num_proj)					
1	CARME	410000	MATARO	1	1
2	EUGENIA	350000	TOLEDO	2	2
3	JOSEP	250000	SITGES	3	1
4	RICARDO	410000	BARCELONA	1	1
11	NURIA	150000	VIC	3	2

■ Dades obtingudes com a resultat de la consulta

## Consultes sobre una taula: Ordenació

```
SELECT <columnes_a_seleccionar> | *  
FROM <taula_a_consultar>  
[ WHERE <condicions> ]  
ORDER BY <columna> [DESC | ASC],.... ;
```

- En el resultat s'obté les dades ordenades segons les columnes que s'explicitin a la clausula **ORDER BY**.
- Si per a una columna no es posa **DESC** s'enten que la classificació, segons els seus valors, es vol que sigui ascendent. Cosa que també es pot demanar explícitament amb la paraula clau **ASC**.

## Consultes sobre una taula: Ordenació - Exemple

```
SELECT num_empl, nom_empl, sou  
FROM empleats  
WHERE num_dpt IN (1,2)  
ORDER BY sou DESC, nom_empl;
```

	num_empl	nom_empl	sou
	1	CARME	410000
	4	RICARDO	410000
	2	EUGENIA	350000
	12	NURIA	150000

	num_empl	nom_empl	sou	ciutat_empl	num_dpt	num_proj
	1	CARME	410000	MATARO	1	1
	2	EUGENIA	350000	TOLEDO	2	2
	3	JOSEP	250000	SITGES	3	1
	4	RICARDO	410000	BARCELONA	1	1
	11	NURIA	150000	VIC	3	2
	12	NURIA	150000	MATARO	1	5

■ Fils que compleixen la condició del WHERE

■ Dades tal com s'obtenen (ordenades) com a resultat de la consulta

## Consultes sobre una taula: Resultats sense repeticions

```
SELECT [ DISTINCT | ALL ] <columnes_a_seleccionar>  
FROM <taula_a_consultar>  
[ WHERE <condicions> ] ;
```

- Si volem que el resultat d'una consulta se'n doni sense repeticions, cal utilitzar la paraula clau **DISTINCT**.
- Si no es posa res se'n donarà el resultat amb repeticions (en cas de que n'hi hagin). Cosa que també es pot demanar explícitament amb la paraula clau **ALL**.

## Consultes sobre una taula: Resultats sense repeticions

```
SELECT DISTINCT nom_empl, sou  
FROM empleats  
WHERE num_dpt IN (1,3);
```

resultat

nom_empl	sou
CARME	410000
JOSEP	250000
RICARDO	410000
NURIA	150000

	num_empl	nom_empl	sou	ciutat_empl	num_dpt	num_proj
	1	CARME	410000	MATARO	1	1
	2	EUGENIA	350000	TOLEDO	2	2
	3	JOSEP	250000	SITGES	3	1
	4	RICARDO	410000	BARCELONA	1	1
	11	NURIA	150000	VIC	3	2
	12	NURIA	150000	MATARO	1	5

■ Fils que compleixen la condició del WHERE

■ Dades obtingudes com a resultat de la consulta

## Consultes sobre una taula: Funcions d'agregació

```
SELECT <funcions_d'agregació>
FROM <taula_a_consultar>
[ WHERE <condicions> ] ;
```

- Són funcions que s'apliquen sobre el conjunt de files de la **taula\_a\_consultar** que compleixen les **condicions** especificades a la clàusula **WHERE**.

### - COUNT:

- COUNT(\*)** - número de files que compleixen la condició del where
- COUNT(DISTINCT <columna>)** – número de valors diferents de la columna, sense comptar valors NULL, per a les files que compleixen la condició del where.
- COUNT(<columna>)** – número de valors de la columna, sense comptar valors NULL, per a les files que compleixen la condició del where.

### - SUM (expressió), MIN(expressió), MAX(expressió), AVG(expressió):

- expressió** pot ser simplement una columna, o pot ser un càlcul a partir del valor de diferents columnes i constants.
- SUM**: dóna la suma dels valors resultants de calcular l'expressió per a les files que compleixen la condició del WHERE
- MIN**: dóna el valor mínim dels resultats de calcular l'expressió per a les files que compleixen la condició del WHERE
- MAX**: dóna el valor màxim dels resultats de calcular l'expressió per a les files que compleixen la condició del WHERE
- AVG**: dóna el valor promig dels resultats de calcular l'expressió per a les files que compleixen la condició del WHERE

## Consultes sobre una taula: Funcions d'agregació - Exemple

```
SELECT COUNT(*) AS quantEmpl ,
COUNT(DISTINCT nom_empl) AS
quantNoms,
SUM(sou*0.1) AS partSou resultat quantEmpl quantNoms partSou
FROM empleats
WHERE num_dpt IN (1,3);
```

	quantEmpl	quantNoms	partSou
	5	4	137000

				<b>num_dpt</b>	<b>num_proj</b>
1	CARME	410000	MATARÓ	1	1
2	EUGENIA	350000	TOLEDO	2	2
3	JOSEP	250000	SITGES	3	1
4	RICARDO	410000	BARCELONA	1	1
11	NURIA	150000	VIC	3	2
12	NURIA	150000	MATARÓ	1	5

■ Files que compleixen la condició del WHERE

■ Resultat de la consulta

## Consultes sobre una taula: Agrupació de files

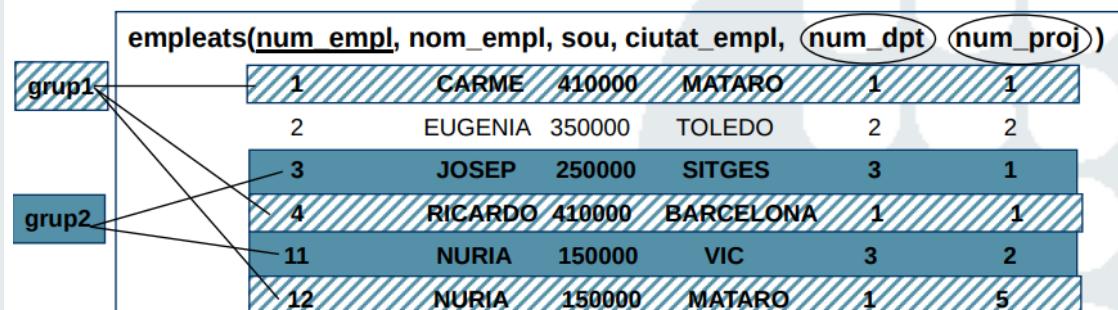
```
SELECT <columnes_a_seleccionar> [,<funcions_d'agregació>]
FROM <taula_a_consultar>
[ WHERE <condicions> ]
GROUP BY <columnes_segons_les_que_agrupar>;
```

- S'organitza en grups les files de la **taula\_a\_consultar** que compleixen les **condicions** especificades a la clàusula **WHERE**, segons el seu valor per les **columnes\_segons\_les\_que\_agrupar**.
- El resultat de la consulta és el valor de les **columnes\_a\_seleccionar** per cadascun dels grups de files obtinguts.
- Les **columnes\_a\_seleccionar** han de ser columnes que continguin el mateix valor per totes les files dins d'un grup.
- En el resultat es pot demanar també el valor de **funcions\_d'agregació** que es calculen per cadascun dels grups de files obtinguts.

## Consultes sobre una taula: Agrupació de files - Exemple

```
SELECT num_dpt,
COUNT(*) AS quantEmpl
FROM empleats
WHERE num_dpt IN (1,3) resultat
GROUP BY num_dpt;
```

<b>num_dpt</b>	<b>quantEmpl</b>
1	3
3	2



■ Files que compleixen la condició del WHERE, agrupades segons indica la clàusula GROUP BY

■ Dades obtingudes com a resultat de la consulta. Hi ha un resultat per cada grup

## Consultes sobre una taula: Condicions sobre grups

```
SELECT <columnes_a_seleccionar> [,<funcions_d'agregació>]
FROM <taula_a_consultar>
[ WHERE <condicions> ]
GROUP BY <columnes_segons_les_que_agrupar>
HAVING <condicions_per_grups>;
```

- En el cas de posar la clàusula **HAVING**, només apareix un resultat per cada grup que compleix les **condicions\_per\_grups**.
- Les **condicions\_per\_grups** seran comparacions entre constants, columnes amb un valor únic per cada grup i valors de funcions d'agregació (també amb un únic valor per cada grup). Per exemple: si s'agrupa els empleats per departament, el sou de cada empleat d'un departament pot ser diferent, però el **MAX** del sou dels empleats d'un departament té valor únic.
- Les **funcions d'agregació** té sentit aplicar-les a columnes que poden tenir valors diferents entre les files que componen els grups.

## Consultes sobre una taula: Condicions sobre grups - Exemple 2

```
SELECT DISTINCT num_dpt
FROM empleats
GROUP BY num_dpt, ciutat_empl
HAVING COUNT(*) >= 2;
```

empleats(num_empl, nom_empl, sou, ciutat_empl, num_dpt, num_proj)						
grup1	1	CARME	410000	MATARO	1	1
grup2	2	EUGENIA	350000	TOLEDO	2	2
grup3	3	JOSEP	250000	SITGES	3	1
grup4	4	RICARDO	410000	BARCELONA	1	1
grup5	11	NURIA	150000	VIC	3	2
	12	NURIA	150000	MATARO	1	5
	13	ALBERT	150000	BARCELONA	1	5

■ ■ ■ En no haver-hi WHERE, són totes les files les que s'agrupen segons indica la clàusula GROUP BY  
 ■ ■ ■ Dades obtingudes com a resultat de la consulta. Un resultat per cada grup que compleix la condició del HAVING, sense resultats repetits degut a la clàusula DISTINCT.

## Consultes sobre una taula: Condicions sobre grups - Exemple 1

```
SELECT num_dpt, SUM(sou) AS sumaSous
FROM empleats
GROUP BY num_dpt
HAVING COUNT(*) >= 3;
```

resultat → num\_dpt      sumaSous

1	970000
---	--------

## Consultes sobre més d'una taula: Format bàsic

```
SELECT <columnes_a_seleccionar> | *
FROM <taules_a_consultar>
[ WHERE <condicions> ] ;
```

- El resultat de la consulta és el valor de les **columnes\_a\_seleccionar** de les **taules\_a\_consultar** únicament per a la fila o files que compleixen les **condicions** especificades a la clausula **WHERE**.
- En el cas de no posar la clausula **WHERE**, el resultat és el valor de les **columnes\_a\_seleccionar** per a les files obtingudes del producte cartesià de les files a les **taules\_a\_consultar**.
- Si posem un \* en lloc de les **columnes\_a\_seleccionar** indica que estem interessats en totes les columnes de les **taules\_a\_consultar**.

■ ■ ■ En no haver-hi WHERE, totes les files agrupades segons indica la clàusula GROUP BY  
 ■ ■ ■ Dades obtingudes com a resultat de la consulta, un resultat per cada grup que compleix la condició del HAVING.

## Consultes sobre més d'una taula: Format bàsic – Exemple 1

**SELECT \***

resultat

FROM empleats e, projectes p;

e.num_empl	e.nom_empl	e.num_proj	p.num_proj	p.nom_proj
1	CARME	1	1	IBDTEL
1	CARME	1	2	IBDVID
1	CARME	1	3	IBDTEF
3	JOSEP	1	1	IBDTEL
3	JOSEP	1	2	IBDVID
3	JOSEP	1	3	IBDTEF

empleats(num\_empl, nom\_empl, num\_proj) projectes(num\_proj, nom\_proj)

1	CARME	1	IBDTEL
3	JOSEP	1	IBDVID

Dades obtingudes com a resultat de la consulta. Cal notar que, de totes les files, les que segurament podem estar interessats, són aquelles marcades en negreta (són aquelles en què el número de projecte en què treballa l'empleat coincideix amb el número de projecte del projecte).

## Consultes sobre més d'una taula: Format bàsic – Exemple 2

**SELECT e.num\_empl, p.num\_proj,**  
p.nom\_proj

FROM empleats e, projectes p

**WHERE e.num\_proj = p.num\_proj;**

resultat

e.num_empl	p.num_proj	p.nom_proj
1	1	IBDTEL
3	1	IBDTEL

empleats(num\_empl, nom\_empl, num\_proj) projectes(num\_proj, nom\_proj)

1	CARME	1	IBDTEL
3	JOSEP	1	IBDVID

Dades obtingudes com a resultat de la consulta. Cal notar que només apareixen aquelles combinacions en les que coincideix el número de projecte en què treballa l'empleat, amb el número de projecte del projecte.

## Consultes sobre més d'una taula: Sintaxis alternativa– Clàusula Inner Join - Exemple 3

Sintaxis alternativa– Clàusula Inner Join - Exemple 3

La condició de combinació de les taules es pot escriure:

- O bé en la clàusula WHERE
- O bé usant la clàusula JOIN en el FROM
  - INNER JOIN requereix la condició de combinació de manera explícita
  - NATURAL INNER JOIN fa la combinació per les columnes amb el mateix nom en les taules implicades.

**SELECT e.num\_empl, p.num\_proj, p.nom\_proj**

FROM empleats e, projectes p

**WHERE e.num\_proj = p.num\_proj;**

**SELECT e.num\_empl, p.num\_proj, p.nom\_proj**

FROM empleats e **INNER JOIN** projectes p **ON** e.num\_proj = p.num\_proj;

**SELECT e.num\_empl, p.num\_proj, p.nom\_proj**

FROM empleats e **NATURAL INNER JOIN** projectes p;

Les tres sentències anteriors són totalment equivalents.

## Consultes sobre varíes taules: Exemple amb agrupació de files

```
SELECT p.num_proj, p.nom_proj
FROM projectes p, empleats e
WHERE p.num_proj = e.num_proj
GROUP BY p.num_proj
HAVING p.pressupost < SUM(e.sou);
```

resultat

p.num\_proj, p.nom\_proj

1 IBDTEL

e.num\_empl, e.nom\_empl, e.sou, e.num\_proj, p.num\_proj, p.nom\_proj, p.pressupost

1	CARME	410000	1	1	IBDTEL	1000000
2	EUGENIA	350000	2	2	IBDVID	500000
3	JOSEP	250000	1	1	IBDTEL	1000000
4	RICARDO	410000	1	1	IBDTEL	1000000
11	NURIA	150000	2	2	IBDVID	500000

Files resultants de la combinació dels empleats amb els projectes, agrupades segons la clàusula GROUP BY

Dades obtingudes com a resultat de la consulta. Un resultat per cada grup que compleix la condició del HAVING. Només hi ha un projecte tal que el seu pressupost és inferior a la suma del sou dels empleats que hi treballen.

Cal notar que l'atribut p.nom\_proj pot estar entre els atributs del select perquè té un valor únic per cada grup. Igualment, l'atribut p.pressupost pot estar en la condició del having perquè té un valor únic per cada grup.

## Consultes: Unió

```
SELECT <columnes_a_seleccionar> | *
FROM <taules_a_consultar>
[ WHERE <condicions> ]
UNION
SELECT <columnes_a_seleccionar> | *
FROM <taules_a_consultar>
[ WHERE <condicions> ]
[ ORDER BY <columna> [DESC|ASC],... ] ;
```

- El resultat és la unió del resultat obtingut de les dues sentències **SELECT**.
- Les **columnes\_a\_seleccionar** en les dues sentències **SELECT** han de ser semànticament compatibles
- Sempre s'obté resultats sense repeticions (en molts SGBDR ja surten ordenats).
- Les columnes que apareixen a clausula **ORDER\_BY** han de ser un subconjunt de les **columnes\_a\_seleccionar** del primer **SELECT**.

## Consultes: Unió - Exemple

```
SELECT ciutat_empl
FROM empleats
UNION
SELECT ciutat_dpt
FROM departaments
ORDER BY ciutat_empl DESC;
```

resultat

ciutat_empl
SITGES
MATARÓ
MADRID
BARCELONA

empleats( <u>num_empl</u> , nom_empl, ciutat_empl)			departaments( <u>num_dpt</u> , ciutat_dpt)		
1	CARME	MATARÓ	1	BARCELONA	
3	JOSEP	SITGES	2	MADRID	
			3	BARCELONA	

Dades obtingudes com a resultat de la consulta.

## Consultes: Diferència

```
SELECT <columnes_a_seleccionar> | *
FROM <taules_a_consultar>
WHERE <columna_taula> NOT IN ( SELECT <columna_a_seleccionar>
                                FROM <taules_a_consultar>
                                [ WHERE <condicions> ]);
```

```
SELECT <columnes_a_seleccionar> | *
FROM <taules_a_consultar>
WHERE NOT EXISTS ( SELECT *
                    FROM <taules_a_consultar>
                    WHERE <condicions> );
```

- Hi ha dos formes alternatives de fer una diferència amb NOT IN o bé amb NOT EXISTS.
- Un NOT IN és cert si el valor de la columna *columna\_taula* no està en el resultat de la subconsulta.
- Un NOT EXISTS és cert si la subconsulta no dóna cap resultat.
- Hi ha altres maneres de fer una diferència (operador Except en SQL estàndard), encara que hi ha diferents noms de l'operador en diferents sistemes.

## Consultes: Diferència – Exemples

```
SELECT p.num_proj, p.nom_proj
FROM projectes p
WHERE p.num_proj NOT IN (SELECT e.num_proj
                           FROM empleats e);
```

```
SELECT p.num_proj, p.nom_proj
FROM projectes p
WHERE NOT EXISTS (SELECT * FROM empleats e
                   WHERE p.num_proj = e.num_proj);
```

p.num\_proj p.nom\_proj

2	IBDVID
3	IBDTEF
4	IBDCOM

empleats( <u>num_empl</u> , nom_empl, <u>num_proj</u> )		projectes( <u>num_proj</u> , nom_proj)	
1	CARME	1	IBDTEL
3	JOSEP	1	IBDVID
		2	IBDTEF
		3	IBDCOM
		4	

- Dades obtingudes com a resultat de la consulta, en qualsevol de les dues alternatives.
  - En qualsevol cas, la consulta dóna aquells projectes que no tenen cap empleat assignat.
- NOT IN: Un projecte és al resultat de la consulta si el seu número no està entre els números de projecte dels empleats.
- NOT EXISTS: Un projecte és al resultat de la consulta si no existeix cap empleat que tingui el seu número de projecte.

## Subconsultes en sentències Delete - Exemple

```
DELETE FROM projectes
WHERE NOT EXISTS (SELECT *
    FROM empleats e
    WHERE e.num_proj = projectes.num_proj);
```

empleats( <u>num_empl</u> , nom_empl, <u>num_proj</u> )		projectes( <u>num_proj</u> , nom_proj)
1	CARME	1
3	JOSEP	1
		1 IBDTEL
		2 IBDVID
		3 IBDTEF
		4 IBDCOM

- Files esborrades com a resultat de la sentència. La sentència elimina aquells projectes que no tenen cap empleat assignat. Cal notar que la subconsulta obté resultats per aquells projectes que tenen com a mínim un empleat.

## Subconsultes en sentències Update - Exemple

```
UPDATE projectes
SET pressupost = pressupost + (pressupost * 0,1)
WHERE 2 <= (SELECT COUNT(*)
    FROM empleats e
    WHERE projectes.num_proj = e.num_proj);
```

empleats( <u>num_empl</u> , nom_empl, <u>num_proj</u> )		projectes( <u>num_proj</u> , pressupost)
1	CARME	1
3	JOSEP	1
		1 1100000
		2 500000
		3 4500000
		4 2000000

- Files modificades com a resultat de la sentència. La sentència puja el pressupost d'aquells projectes que tenen dos o més empleats assignats. Cal notar que la subconsulta el que fa és calcular el nombre d'empleats del projecte que s'està considerant si cal modificar o no.

## Subconsultes en sentències Select – Exemple 2

```
SELECT p.num_proj, p.nom_proj
FROM projectes p
WHERE p.pressupost < (SELECT SUM(e.sou)
    FROM empleats e
    WHERE e.num_proj=p.num_proj);
```

resultat → p.num\_proj,p.nom\_proj

1	IBDTEL
---	--------

projectes(num\_proj,nom\_proj,p.pressupost)    empleats(num\_empl, nom\_empl, sou, num\_proj)

1	IBDTEL	1000000	1 CARME 410000 1
2	IBDVID	500000	2 EUGENIA 350000 2
3			3 JOSEP 250000 1
4			4 RICARDO 410000 1
11			11 NURIA 150000 2

□ Dades obtingudes com a resultat de la sentència. La sentència dóna els projectes que tenen un pressupost inferior a la suma del sou dels empleats que hi estan assignats. Cal notar que la subconsulta retorna la suma del sou dels empleats assignats al projecte que s'està considerant.

■ Files seleccionades en la subconsulta quan es considera el projecte número 1. La suma dels sous és 1070000.

■ Files seleccionades en la subconsulta quan es considera el projecte número 2. La suma dels sous és 500000

## Subconsultes en sentències Insert – Exemple

```
INSERT INTO clients
(SELECT num_empl,nom_empl,200000
FROM empleats
WHERE num_dpt IN (2,3));
```

clients(num\_client, nom, credit)    empleats(num\_empl, nom\_empl, num\_dpt )

2	EUGENIA	200000	1 CARME 1
3	JOSEP	200000	2 EUGENIA 2
			3 JOSEP 3
			4 RICARDO 1

■ Files seleccionades a la subconsulta.

■ Files inserides a la taula clients.

Doneu una sentència SQL per obtenir els números i els noms dels departament situats a MADRID, que tenen empleats que guanyen més de 200000. Pel joc de proves que trobareu al fitxer adjunt, la sortida ha de ser:

NUM_DPT	NOM_DPT
5	VENDES

[Fitxer adjunt](#)

Solució:

```
SELECT DISTINCT D.NUM_DPT, D.NOM_DPT
FROM DEPARTAMENTOS D, EMPLEATS E
WHERE D.CIUTAT_DPT = 'MADRID' AND E.SOU > 200000 AND D.num_dpt = E.num_dpt;
```

Doneu una sentència SQL per obtenir el nom del departament on treballa i el nom del projecte on està assignat l'empleat número 2. Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

Nom_dpt	Nom_proj
MARKETING	IBDVID

[Fitxer adjunt](#)

Solució:

```
select D.NOM_DPT, P.nom_proj
from departaments D, empleats E, projectes P
where E.num_empl=2 and E.num_proj=P.num_proj and E.num_dpt=D.num_dpt;
```

Obtenir per cada departament situat a MADRID la mitjana dels sous dels seus empleats. Concretament, cal donar el número de departament, el nom de departament i la mitjana del sou.

Pel joc de proves que trobareu al fitxer adjunt, la sortida ha de ser:

NUM_DPT	NOM_DPT	SOU
5	VENDES	250000

[Fitxer adjunt](#)

Solució:

```
select D.NUM_DPT, D.nom_dpt, AVG(E.SOU) as mitjana_Sou
from departaments D natural inner JOIN empleats E
where D.ciutat_dpt = 'MADRID'
group by D.num_dpt;
```

Doneu una sentència SQL per obtenir el nom dels empleats que guanyen el sou més alt. Cal ordenar el resultat **descendentment** per nom de l'empleat.

Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

NOM_EMPL
JOAN

[Fitxer adjunt](#)

Solució:

```
SELECT distinct e.nom_empl
FROM empleats e
WHERE e.sou = ( SELECT max(e1.sou) FROM empleats e1)
order by e.nom_empl desc;
```

## Subconsultes en clàusules Having – Exemple

```
SELECT d.num_dpt, d.nom_dpt, 100*SUM(e.sou)/d.pressupost AS percSous
FROM departaments d, empleats e
WHERE d.num_dpt = e.num_dpt
GROUP BY d.num_dpt
HAVING SUM(e.sou) > ( SELECT SUM(e1.sou) → resultat
FROM empleats e1
WHERE e1.num_dpt = 3);
```

d.num_dpt	d.nom_dpt	percSous
1	DIRECCIO	82
2		
3		
11		

empleats(num_empl, nom_empl, sou, num_dpt)			departaments(num_dpt, nom_dpt, pressupost)		
1	CARME	410000	1	DIRECCIO	1000000
2	EUGENIA	350000	2	DIRECCIO	2000000
3	JOSEP	350000	3	MARQUETING	2500000
4	RICARDO	410000	1		
11	NURIA	150000	3		

Dades obtingudes com a resultat de la sentència. La sentència dóna els departaments tals que la suma del sou dels seus empleats és superior a la suma del sou dels empleats del departament número 3. Cal notar que la subconsulta retorna la suma del sou dels empleats del departament número 3.

Files seleccionades en la subconsulta són les que són del departament que es considera. La suma dels soués 500000.

Doneu una sentència SQL per obtenir els números i els noms dels projectes que tenen assignats dos o més empleats.

Cal ordenar el resultat **descendentement** per número de projecte.

Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

NUM_PROJ	NOM_PROJ
1	IBDTEL

[Fitxer adjunt](#)

Solució:

```
select distinct p.num_proj, p.nom_proj
from projectes p
where 2 <= (select count(*) from empleats e where e.num_proj = p.num_proj)
order by p.num_proj desc;
```

Doneu una sentència SQL per incrementar en 500000 el pressupost dels projectes que tenen algun empleat que treballa a BARCELONA.

Pel joc de proves que trobareu al fitxer adjunt, el pressupost del projecte que hi ha d'haver després de l'execució de la sentència és 1100000

[Fitxer adjunt](#)

Solució:

```
UPDATE projectes
SET pressupost = pressupost + 500000
where 1 <= (select count(*)
            from departaments d, empleats e
            where (projectes.num_proj = e.num_proj and e.num_dpt = d.num_dpt and d.ciutat_dpt = 'BARCELONA'));
```

Doneu una sentència SQL per obtenir la quantitat d'empleats que treballen a la ciutat de MADRID.

Pel joc de proves que trobareu al fitxer adjunt, la sortida ha de ser:

EMPL_MADRID
1

[Fitxer adjunt](#)

Solució:

```
select count(*) as EMPL_MADRID
from empleats e, departaments d
where e.num_dpt = d.num_dpt and d.ciutat_dpt = 'MADRID';
```

Obtenir per cada departament situat a MADRID quin és el sou més gran. Concretament, cal llistar el número de departament i el sou més gran. El resultat ha d'estar ordenat ascendentment per número de departament.

Pel joc de proves que trobareu al fitxer adjunt, la sortida ha de ser:

NUM_DPT	SOU
5	250000

[Fitxer adjunt](#)

Solució:

```
SELECT e.num_dpt, MAX(e.sou) AS SOU
FROM empleats e
INNER JOIN departaments d ON e.num_dpt = d.num_dpt
WHERE d.ciutat_dpt = 'MADRID'
GROUP BY e.num_dpt
ORDER BY e.num_dpt ASC;
```

Doneu una sentència SQL per obtenir el número i el nom dels departaments que no tenen cap empleat que visqui a MADRID.

Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

NUM_DPT	NOM_DPT
3	MARKETING

[Fitxer adjunt](#)

Solució:

```
select d.num_dpt, d.nom_dpt
from departaments d
where not exists (select * from empleats e where e.num_dpt = d.num_dpt and
e.ciutat_empl='MADRID');
```

Doneu una sentència SQL per obtenir les ciutats on hi viuen empleats però no hi ha cap departament.

Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

CIUTAT_EMPL
GIRONA

[Fitxer adjunt](#)

Solució:

```
select distinct e.ciutat_empl
from empleats e
where not exists (select * from departaments d where e.ciutat_empl = d.ciutat_dpt);
```

Doneu una sentència SQL per obtenir el número i nom dels departaments que tenen dos o més empleats que viuen a ciutats diferents.

Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

NUM_DPT	NOM_DPT
3	MARKETING

[Fitxer adjunt](#)

Solució:

```
select d.num_dpt, d.nom_dpt
from departaments d
where 2 <= (select count(distinct e.ciutat_empl) from empleats e where e.num_dpt =
d.num_dpt);
```

Tenint en compte l'esquema de la BD que s'adjunta, proposeu una sentència de creació de les taules següents:

comandes(numComanda, instantComanda, client, encarregat, supervisor)

productesComprats(numComanda, producte, quantitat, preu)

La taula *comandes* conté les comandes fetes.

La taula *productesComprats* conté la informació dels productes comprats a les comandes de la taula *comandes*.

En la creació de les taules cal que tingueu en compte que:

- No hi poden haver dues comandes amb un mateix número de comanda.
- Un client no pot fer dues comandes en una mateix instant.
- L'encarregat és un empleat que ha d'existeix necessàriament a la base de dades, i que té sempre tota comanda.
- El supervisor és també un empleat de la base de dades i que s'assigna a algunes comandes en certes circumstàncies.
- No hi pot haver dues vegades un mateix producte en una mateixa comanda. Ja que en cas de el client compri més d'una unitat d'un producte en una mateixa comanda s'indica en l'atribut quantitat.
- Un producte sempre s'ha comprat en una comanda que ha d'existeix necessàriament a la base de dades.
- La quantitat de producte comprat en una comanda no pot ser nul, i té com a valor per defecte 1.
- Els atributs numComanda, instantComanda, quantitat i preu són de tipus *integer*.
- Els atributs client, producte són *char(30)*, i *char(20)* respectivament.
- L'atribut instantComanda no pot tenir valors nuls.

Respecteu els noms i l'ordre en què apareixen les columnes (fins i tot dins la clau o claus que calgui definir). Tots els noms s'han de posar en majúscules/minúscules com surt a l'enunciat.

#### create table comandes

```
(    numComanda integer,  
    instantComanda integer not null,  
    client char(30),  
    encarregat integer not null,  
    supervisor integer,  
    primary key (numComanda),  
    unique (instantComanda, client),  
    foreign key (encarregat) references empleats (num_empl),  
    foreign key (supervisor) references empleats (num_empl));
```

Doneu una sentència SQL per obtenir el nom dels professors que o bé se sap el seu número de telèfon (valor diferent de null) i tenen un sou superior a 2500, o bé no se sap el seu número de telèfon (valor null) i no tenen cap assignació a un despatx amb superfície inferior a 20.

Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

NomProf  
toni

Fitxer adjunt

Solució:

```
select p.nomprof  
from professors p  
where ((p.telefon is not null and p.sou > 2500) or (p.telefon is null and not exists(select * from  
assignacions a, despatxos d where p.dni=a.dni and a.numero=d.numero and a.modul=d.modul  
and d.superficie < 20)));
```

#### create table productesComprats

```
(    numComanda integer,  
    producte char(20),  
    quantitat integer not null default 1,  
    preu integer,  
    primary key (numComanda, producte),  
    foreign key (numComanda) references comandes (numComanda));
```

Donar una sentència SQL per obtenir els professors que tenen alguna assignació finalitzada (instantFi different de null) a un despatx amb superfície superior a 15 i que cobren un sou inferior o igual a la mitjana del sou de tots els professors. En el resultat de la consulta ha de sortir el dni del professor, el nom del professor, i el darrer instant en què el professor ha estat assignat a un despatx amb superfície superior a 15.

Pel joc de proves que trobareu al fitxer adjunt, la sortida ha de ser:

DNI NomProf Darrer\_instant

111 toni 344

#### Fitxer adjunt

Solució:

```
select p.dni, p.nomprof, max(a.instantfi)
from professors p, assignacions a, despatxos d
where ((a.instantfi is not null)
and (a.dni = p.dni)
and a.numero = d.numero
and a.modul = d.modul
and d.superficie > 15)
and p.sou <= (select avg(sou) from professors))
group by p.dni;
```

Suposem la base de dades que podeu trobar al fitxer adjunt.

Suposem que aquesta base de dades està en un estat on no hi ha cap fila.

Doneu una seqüència de sentències SQL d'actualització (INSERTs i/o UPDATEs) sobre la taula que assignacions que violi la integritat referencial de la clau forana que referencia la taula Despatxos. Les sentències **NOMÉS** han de violar aquesta restricció.

#### Fitxer adjunt

Solució:

```
insert into professors values('111', 'Elias', 987654,999999);
insert into assignacions values ('111', 'OMEGA','118',109,344);
```

Doneu una seqüència d'operacions d'algebra relacional per obtenir el nom del departament on treballa i el nom del projecte on està assignat l'empleat número 2.

Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

Nom\_dpt Nom\_proj

MARKETING IBDVID

#### Fitxer adjunt

Solució:

```
A = empleats(num_empl = 2)
R =A * projectes
C = R * departaments
D = C[nom_dpt, nom_proj]
```

Suposeu la base de dades que podeu trobar al fitxer adjunt.

Doneu una seqüència de sentències SQL d'actualització (INSERTs i/o UPDATEs) de tal manera que, un cop executades, el resultat de la consulta següent sigui el que s'indica. **El nombre de files de cada taula ha de ser el més petit possible, i hi ha d'haver com a màxim un professor.**

Per a la consulta:

```
Select count(*) as quant  
From assignacions ass  
Where ass.instantInici>50  
Group by ass.instantInici  
order by quant;
```

El resultat haurà de ser:

```
quant  
1  
2
```

[Fitxer adjunt](#)

Solució:

```
insert into professors values ('111', 'juan carlos', 64672878, 3000);  
insert into despatxos values ('A5', '34', 50);  
insert into despatxos values ('omega', '108', 25);  
insert into assignacions values ('111', 'A5', '34', 55, null);  
insert into assignacions values ('111', 'omega', '108', 55, null);  
  
insert into assignacions values ('111', 'omega', '108', 60, null);
```

Doneu una seqüència d'operacions d'algebra relacional per obtenir el número i nom dels departaments tals que tots els seus empleats viuen a MADRID. El resultat no ha d'incloure aquells departaments que no tenen cap empleat.

Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

Num_dpt	Nom_dpt
3	MARKETING

[Fitxer adjunt](#)

Solució:

```
A = empleats(ciutat_empl <> 'MADRID')  
B = departaments*A  
C=B[num_dpt, nom_dpt, planta, edifici, ciutat_dpt]  
F = departaments*empleats  
G = F[num_dpt, nom_dpt, planta, edifici, ciutat_dpt]
```

D = G-C

E = D[num\_dpt, nom\_dpt]

Doneu una seqüència d'operacions de l'àlgebra relacional per obtenir el número i nom dels departaments que tenen dos o més empleats que viuen a ciutats diferents.

Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

Num_dpt	Nom_dpt
3	MARKETING

[Fitxer adjunt](#)

Solució:

```
A = empleats[ciutat_empl, num_dpt,num_empl]
B = A {ciutat_empl ->ciutat_empl1, num_dpt -> num_dpt1, num_empl -> num_empl1}
C = A[num_dpt = num_dpt1, ciutat_empl <> ciutat_empl1]B
D = C*departaments
E = D[num_dpt,nom_dpt]
```

Donar una seqüència d'operacions d'àlgebra relacional per obtenir informació sobre els despatxos que només han estat ocupats per professors amb sou igual a 100000. Es vol obtenir el modul i el numero d'aquests despatxos.

Pel joc de proves que trobareu al fitxer adjunt, la sortida ha de ser:

Modul	Numero
Omega	128

[Fitxer adjunt](#)

Solució:

```
A = professors(sou=>100000)
B = A*assignacions
C = B[modul, numero]
D = assignacions[modul, numero]
E = D-C
```

Doneu una sentència SQL per obtenir el número i nom dels departaments que tenen 2 o més empleats que viuen a la mateixa ciutat.

Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

NUM_DPT	NOM_DPT
3	MARKETING

[Fitxer adjunt](#)

Solució:

```
select d.num_dpt, d.nom_dpt
from departaments d
WHERE d.num_dpt IN (
    SELECT e.num_dpt
    FROM empleats e
    WHERE e.num_dpt = d.num_dpt
    GROUP BY e.ciutat_empl
    HAVING COUNT(*) >= 2
);
```

Doneu una sentència SQL per esborrar els departaments que no tenen cap empleat.

Pel joc de proves que trobareu al fitxer adjunt, els departaments que hi ha d'haver a la taula departaments després de l'execució de la sentència són els següents:

NUM\_DPT

3

[Fitxer adjunt](#)

Solució:

```
DELETE FROM departaments  
WHERE num_dpt NOT IN (SELECT DISTINCT num_dpt FROM empleats);
```

Doneu una sentència d'inserció de files a la taula *cost\_ciutat* que l'ompli a partir del contingut de la resta de taules de la base de dades. Tingueu en compte el següent:

Hi haurà una fila de la taula per cada ciutat on hi ha un departament. El valor de l'atribut cost serà la suma del sou dels empleats dels departaments situats a la ciutat.

Només han de sortir les ciutats on hi ha departament que tinguin empleats.

Pel joc de proves públic del fitxer adjunt, un cop executada la sentència d'inserció, a la taula *cost\_ciutat* hi haurà les tuples següents:

CIUTAT\_DPT COST

BARCELONA 100

[Fitxer adjunt](#)

Solució:

```
INSERT INTO cost_ciutat (ciutat_dpt, cost)  
SELECT DISTINCT d.ciutat_dpt, SUM(e.sou) as cost  
FROM departaments d  
INNER JOIN empleats e ON d.num_dpt = e.num_dpt  
GROUP BY d.ciutat_dpt;
```

Doneu una seqüència d'operacions d'àlgebra relacional per obtenir el nom dels empleats que guanyen més que l'empleat amb num\_empl 3.

Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

Nom\_empl

-----

JOAN

PERE

[Fitxer adjunt](#)

Solució:

```
A = empleats(num_empl = 3)  
B = A{sou -> sou1, nom_empl -> nom_empl1}  
R = B[nom_empl1,sou1]  
C = empleats(num_empl <> 3)  
D = C[nom_empl,sou]
```

E = R[sou1 < sou]D

F = E[nom\_empl]

Donar una sentència SQL per obtenir per cada mòdul on hi hagi despatxos, la suma de les durades de les assignacions finalitzades (instantFi different de null) a despatxos del mòdul. El resultat ha d'estar ordenat ascendentment pel nom del mòdul.

Pel joc de proves que trobareu al fitxer adjunt, la sortida ha de ser:

MODUL	SUMAA
Omega	235

[Fitxer adjunt](#)

Solució:

```
select a.modul, (sum(a.instantFi)-sum(a.instantInici)) as SUMAA
from assignacions a
where a.instantFi is not null
group by a.modul
order by a.modul asc;
```

Doneu una sentència SQL per obtenir els departaments tals que tots els empleats del departament estan assignats a un mateix projecte.

No es vol que surtin a la consulta els departaments que no tenen cap empleat.

Es vol el número, nom i ciutat de cada departament.

Cal resoldre l'exercici **sense fer servir funcions d'agregació**.

Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

Num_dpt	Nom_dpt	Ciutat_dpt
1	DIRECCIO	BARCELONA

[Fitxer adjunt](#)

Solució:

```
select distinct d.num_dpt, d.nom_dpt, d.ciutat_dpt
from departaments d, empleats e
where d.num_dpt = e.num_dpt
and not exists (
    select *
        from empleats e1
        where e1.num_dpt = e.num_dpt
        and e1.num_proj <> e.num_proj);
```

Doneu una seqüència d'operacions en àlgebra relacional per obtenir el nom dels professors que o bé tenen un sou superior a 2500, o bé que cobren menys de 2500 i no tenen cap assignació a un despatx amb superfície inferior a 20.

Pel joc de proves que trobareu al fitxer adjunt, la sortida seria:

NomProf
toni

[Fitxer adjunt](#)

Solució:

A = professors(sou>2500)	F = B[dni]
B = professors(sou<2500)	G = F-E
C = despatxos(superficie<20)	H = G*B
D = C*assignacions	I = A_u_H
E = D[dni]	J = I[nomprof]

Doneu una sentència d'inserció de files a la taula **cost\_ciutat** que l'ompli a partir del contingut de la resta de taules de la base de dades. Tingueu en compte el següent:

Cal inserir una fila a la taula **cost\_ciutat** per cada ciutat on hi ha un o més departaments, però no hi ha cap departament que tingui empleats.

Per tant, només s'han d'inserir les ciutats on cap dels departaments situats a la ciutat tinguin empleats.

El valor de l'atribut cost ha de ser 0.

Pel joc de proves públic del fitxer adjunt, un cop executada la sentència d'inserció, a la taula **cost\_ciutat** hi haurà les tuples següents:

CIUTAT_DPT	COST
BARCELONA	0

[Fitxer adjunt](#)

Solució:

```
insert
into cost_ciutat (ciutat_dpt, cost)
select distinct d.ciutat_dpt, 0 as cost
from departaments d
where not exists
(select *
 from empleats e, departaments d2
 where e.num_dpt = d2.num_dpt and d.ciutat_dpt = d2.ciutat_dpt);
```

Tenint en compte l'esquema de la BD que s'adjunta, proposeu una sentència de creació de la taula següent:

```
presentacioTFG(idEstudiant, titolTFG, dniDirector, dniPresident, dniVocal, instantPresentacio, nota)
```

Hi ha una fila de la taula per cada treball final de grau (TFG) que estigui pendent de ser presentat o que ja s'hagi presentat.

En la creació de la taula cal que tingueu en compte que:

- No hi pot haver dos TFG d'un mateix estudiant.
- Tot TFG ha de tenir un títol.
- No hi pot haver dos TFG amb el mateix títol i el mateix director.
- El director, el president i el vocal han de ser professors que existeixin a la base de dades, i tot TFG té sempre director, president i vocal.
- El director del TFG no pot estar en el tribunal del TFG (no pot ser ni president, ni vocal).
- El president i el vocal no poden ser el mateix professor.
- L'identificador de l'estudiant i el títol del TFG són chars de 100 caràcters.
- L'instant de presentació ha de ser un enter diferent de nul.
- La nota ha de ser un enter entre 0 i 10.
- La nota té valor nul fins que s'ha fet la presentació del TFG.

Respecteu els noms i l'ordre en què apareixen les columnes (fins i tot dins la clau o claus que calgui definir). Tots els noms s'han de posar en majúscues/minúscules com surt a l'enunciat.

[Fitxer adjunt](#)

Solució:

```
create table presentacioTFG
(idEstudiant char(100),
titolTFG char(100) not null,
dniDirector char(50) not null,
dniPresident char(50) not null,
dniVocal char(50) not null,
instantPresentacio integer not null,
nota integer default NULL
check (nota <= 10 and nota >= 0),
primary key(idEstudiant),
unique(titolTFG, dniDirector),
foreign key(dniDirector) references professors,
foreign key(dniPresident) references professors,
foreign key(dniVocal) references professors,
check (dniDirector <> dniPresident and dniDirector <> dniVocal and dniPresident <> dniVocal))
```

## Paràmetres: Retorn d'una única tupla

```
CREATE FUNCTION nom_proc(param_entrada tipus) RETURNS tipus_retorn AS $$  
.....  
RETURN variable_retorn;  
END;  
$$LANGUAGE plpgsql;
```

Nom de la variable que té el valor que retorna la funció

Paràmetre d'entrada: nom i tipus

Tipus de retorn de la funció  
En cas de no retornar res es posa "void"

## Paràmetres: Retorn d'un conjunt de tuples

```
CREATE FUNCTION nom_proc(param_entrada tipus) RETURNS SETOF tipus AS $$  
.....  
RETURN NEXT ciutat_client;  
.....  
END;  
$$LANGUAGE plpgsql;
```

- Cal usar la clàusula RETURN NEXT. Aquesta clàusula no acaba el procediment, sinó que va retornant a cada execució els valors de la variable. El procediment acaba quan s'executa un RETURN sense NEXT, o quan s'arriba al final.
- Per retornar un conjunt de tuples cal utilitzar SETOF quan especificuem el tipus que retorna la funció.

## Paràmetres: Retorn d'una única tupla - Exemple

Aquest procediment obté la ciutat on viu el client amb el DNI que es passa com a paràmetre d'entrada.

```
CREATE FUNCTION trobar_ciutat(dni_client varchar(9))  
RETURNS varchar(15) AS $$  
DECLARE  
    ciutat_client varchar(15);  
BEGIN  
    SELECT ciutat INTO ciutat_client  
    FROM clients  
    WHERE dni=dni_client;  
  
    RETURN ciutat_client;  
END;  
$$LANGUAGE plpgsql;
```

DNI del client

Tipus que tindrà la variable de retorn

Variable de retorn de la ciutat on viu el client

## Paràmetres: Retorn d'un conjunt de tuples - Exemple

Aquest procediment retorna una tupla per cada enter que hi ha entre 0 i el valor del paràmetre d'entrada MAX.

```
CREATE FUNCTION exemple_retorn_n_tuples(max integer)  
RETURNS SETOF integer AS $$  
DECLARE  
    i integer := 0;  
BEGIN  
    LOOP  
        i:=i+1;  
        RETURN NEXT i;  
        EXIT WHEN i = max;  
    END LOOP;  
    RETURN;  
END;  
$$LANGUAGE plpgsql;
```

SETOF del tipus que tindrà la variable de retorn

RETURN NEXT que s'invoca tantes vegades com tuples es vol retornar

## Variables: Declaració

- El valor d'una variable s'emmagatzema en memòria volàtil i per tant, no són considerades objectes de la BD
- Totes les variables definides dins d'un procediment són variables locals.
- L'àmbit de visibilitat d'una variable local queda restringit al procediment a on s'hagi definit
- Sintaxis:  

```
Nom_variable [CONSTANT] type [NOT NULL] [{DEFAULT | :=}expression];
```

```
DECLARE  
nom_client char(15);  
carrer varchar(20) not null;  
edat integer default 18;  
num constant integer default 0;  
dni_client clients.dni%TYPE;
```

Podem utilitzar els mateixos tipus de dades que els utilitzats a les columnes d'una taula.

És possible especificar que el tipus de dades d'una variable és idèntic al tipus de dades d'una determinada columna d'una taula mitjançant la clàusula TYPE.

- Si no s'inicialitzen les variables, per defecte prenen valor NULL

## Variables: Utilització

- Bàsicament, és possible utilitzar variables dins d'un procediment emmagatzemat en les situacions següents:

- En sentències SQL

```
CREATE FUNCTION....  
DECLARE  
    dni_client clients.dni%TYPE;  
    ciutat_client varchar(15);  
BEGIN  
    ....  
    SELECT ciutat INTO ciutat_client  
    FROM clients  
    WHERE dni=dni_client;  
    ...  
END;
```

- En sentències de PL/PGSQL per

- Assignar-hi valors
- Calcular valors
- Controlar el flux d'execució d'un procediment

## Variables: Creació de nous tipus

En alguns casos, com per exemple quan un procediment ha de retornar tuples amb un conjunt d'atributs ens cal definir un nou tipus.

```
CREATE TYPE tipusAdressa AS (  
    carrer varchar(20),  
    num_carrer varchar(4),  
    ciutat varchar (15));
```

```
CREATE FUNCTION exNousTipus()  
RETURNS tipusAdressa AS $$  
DECLARE  
    adressa tipusAdressa;  
    carrer varchar(20);  
BEGIN  
    ....  
    adressa.ciutat:='Badalona';  
    ....  
    carrer := adressa.carrer;  
    ....  
    RETURN adressa;  
END;  
$$ LANGUAGE plpgsql;
```

Creació prèvia al procediment

Declaració d'una variable del tipus

Utilització i accés dels diferents valors de la variable

## Variables: Creació de nous tipus - Exemple

Obtenir l'adreça (concretament el carrer, num\_carrer i ciutat) d'un client

```
CREATE TYPE TAdressa AS (  
    carrer varchar(20),  
    num_carrer varchar(4),  
    ciutat varchar(15)  
);  
  
CREATE FUNCTION trobar_adressa_client (dni_client clients.dni%type)  
RETURNS TAdressa AS $$  
DECLARE  
    dadesCli TAdressa;  
BEGIN  
    SELECT carrer,num_carrer,ciutat INTO dadesCli  
    FROM clients  
    WHERE dni=dni_client;  
  
    RETURN dadesCli;  
END;  
$$LANGUAGE plpgsql;  
  
select * FROM trobar_adressa_client('45678900');
```

## Variables: Assignacions de valors

Només té sentit per sentències o procediments que només retornen una fila:

- Sentència d'assignació de PL/PGSQL

```
numComclient := (SELECT num_com  
                      FROM clients  
                     WHERE dni=dni_client);
```

- Sentència SELECT ... INTO de l'SQL

```
SELECT ciutat INTO ciutat_client  
      FROM clients  
     WHERE dni=dni_client;
```

- Assignar a una variable el que retorna un procediment:

```
imp_comanda := import_una_com(numero_com);  
o bé  
select * from import_una_com(numero_com) into imp_comanda;
```

## Sentències condicionals - Exemple

Obté el descompte del client amb el DNI que es passa per paràmetre.  
Aquest descompte depèn del nombre de comandes del client.

```
CREATE FUNCTION calcul_desc_client(dni_client clients.dni%type)  
RETURNS integer AS $$  
DECLARE  
    descompte INTEGER;  
    qttComClient INTEGER;  
BEGIN  
    IF (EXISTS (SELECT * FROM clients WHERE dni=dni_client))THEN  
        qttComClient:=(SELECT qtt_com FROM clients WHERE dni=dni_client);  
        IF (qttComClient=0) THEN descompte:=0;  
        ELSIF (qttComClient<5) THEN descompte:=1;  
        ELSIF (qttComClient<10) THEN descompte:=3;  
        ELSIF (qttComClient<15) THEN descompte:=5;  
        ELSE descompte:=10;  
        END IF;  
    END IF;  
    RETURN descompte;  
END;  
$$LANGUAGE plpgsql;
```

Execució de la funció:

A screenshot of a PostgreSQL terminal window. The command entered is "select \* from calcul\_desc\_client('35678111') as descompte;". The output shows a single row with the column 'descompte' containing the value '3'. The terminal interface includes tabs for 'Panel de Salida', 'Salida de datos', 'Comentar', 'Mensajes', and 'Historial'.

descompte	integer
1	3

## Sentències condicionals

- La sentència IF serveix per a establir condicions en el flux d'execució d'un procediment:

```
IF condició THEN bloc de sentències  
ELSE bloc de sentències  
END IF;
```

- Podem establir diferents nivells d'aniuament mitjançant la clàusula IF ... THEN ... ELSEIF ... THEN ... ELSE...END IF;

- Condicions:

- Per especificar les condicions podem utilitzar:
  - Operadors lògics: AND, OR, NOT
  - Operadors de comparació: =, <, <=, >, >=, etc
  - Predicats propis d'SQL: BETWEEN, IN, IS NULL, EXISTS
  - Variable PL/pgSQL: FOUND
  - Consultes SQL

## Sentències condicionals – Variable FOUND

La variable FOUND és de tipus booleà.

FOUND té en principi el valor False.

El seu valor pot canviar quan s'executen les sentències següents:

- Una sentència SELECT ... INTO posa FOUND a True si el select obté una fila, i a False si no s'obté cap fila.
- Una sentència UPDATE, INSERT o DELETE posa FOUND a True si com a mínim una fila es veu afectada per la sentència, i a False si no queda afectada cap fila.
- Una sentència FOR. Dintre de cada iteració del FOR, el valor de la variable pot canviar segons les sentències que s'hi executen. Però en sortir del FOR és posa FOUND a True si s'ha iterat una o més vegades, sinó es posa a False.

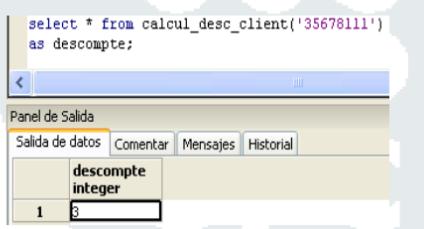
FOUND és una variable local en un procediment. Qualsevol canvi en aquesta variable afecta només al procediment on aquest canvi es produeix.

## Sentències condicionals – Variable FOUND - Exemple

Obté el descompte del client amb el DNI que es passa per paràmetre. Aquest descompte depèn del nombre de comandes del client.

```
CREATE FUNCTION calcul_desc_client(dni_client clients.dni%type)
RETURNS integer AS $$

DECLARE
    descompte INTEGER;
    qttComClient INTEGER;
BEGIN
    SELECT qtt_com into qttComClient FROM clients WHERE dni=dni_client;
    IF FOUND THEN
        IF (qttComClient=0) THEN descompte:=0;
        ELSIF (qttComClient<5) THEN descompte:=1;
        ELSIF (qttComClient<10) THEN descompte:=3;
        ELSIF (qttComClient<15) THEN descompte:=5;
        ELSE descompte:=10;
    END IF;
    END IF;
    RETURN descompte;
END;
$$LANGUAGE plpgsql;
```



## Sentències iteratives: Exemple utilització FOR (1)

Obté les dades de tots els clients d'una determinada ciutat. Indica amb una "P" els clients que són preferents (import total de comandes superior a 60000).

```
CREATE TYPE Tdades_client_tip AS (
    dni_client VARCHAR(9),
    nom_client VARCHAR(15),
    cognom1 VARCHAR(15),
    pref CHAR(1));

CREATE FUNCTION clients_ciutat_tip(ciutat_client clients.ciutat%type)
RETURNS SETOF Tdades_client_tip AS $$
DECLARE dades_clients Tdades_client_tip;
BEGIN
    FOR dades_clients IN SELECT dni,nom,cognom1
        FROM clients
        WHERE ciutat=ciutat_client LOOP
        dades_clients.pref := es_preferent(dades_clients.dni_client);
    END LOOP;
    RETURN;
END;
$$LANGUAGE plpgsql;
```

Procediment que comprova si un client és preferent

## Sentències iteratives FOR, WHILE i LOOP

### Sentència FOR

- S'utilitza habitualment per iterar sobre el conjunt de tuples retornades per una consulta SQL.

```
FOR target IN query
LOOP statements END LOOP;
```

- Es pot utilitzar també quan sabem a priori el nombre d'iteracions a executar.

```
FOR name IN [ REVERSE ] expression .. expression
LOOP statements END LOOP;
```

### Sentències WHILE i LOOP.

- S'utilitzen per definir bucles on el seu acabament estigu definit per una expressió condicional.

#### Sentència LOOP :

```
LOOP statements EXIT [ WHEN expression ]; statements; END LOOP;
```

#### Sentència WHILE:

```
WHILE expression LOOP statements END LOOP;
```

## Sentències iteratives: Exemple utilització FOR (2)

Modifica l'import de la comanda a la taula comandes per cadascuna de les comandes d'un determinat client.

```
CREATE FUNCTION import_totes_com(dni_client clients.dni%type)
RETURNS void AS $$
DECLARE
    num_comanda comandes.num_com%type;
    import comandes.import_total%type;
BEGIN
    FOR num_comanda IN SELECT num_com FROM comandes
        WHERE dni=dni_client
    LOOP
        SELECT SUM(ic.quantitat*i.preu_unitat) INTO import
        FROM items_comanda ic, items i
        WHERE i.num_item=ic.num_item AND ic.num_com=num_comanda;
        UPDATE comandes
        SET import_total=import
        WHERE num_com=num_comanda;
    END LOOP;
END;
$$LANGUAGE plpgsql;
```

## Sentències iteratives: Exemple utilització WHILE

```
CREATE FUNCTION incrementar_preu()RETURNS void as $$  
BEGIN  
    WHILE EXISTS(SELECT * FROM items WHERE preu_unitat<25) LOOP  
        UPDATE items  
        SET preu_unitat=preu_unitat+5  
        WHERE preu_unitat<25;  
    END LOOP;  
END;  
$$LANGUAGE plpgsql;
```

Contingut de la taula Items abans i després d'executar el procediment.

num_item	preu_unitat
1	15
2	20
3	15
4	20
5	15
6	50
7	40

num_item	preu_unitat
1	25
2	25
3	25
4	25
5	25
6	50
7	40

Que hauria passat si en comptes d'utilitzar un WHILE haguéssim utilitzat un FOR ?

## Gestió d'errors

Quan es produeix un error dintre d'un procediment podem:

- **No capturar-lo:** El procediment falla i es retorna l'error concret al nivell superior (JDBC, editor d'SQL, a un altre procediment..)
- **Capturar-lo:**
  - **Opció 1.** El procediment falla i es retorna una **excepció** determinada pel programador al nivell superior.
  - **Opció 2.** El procediment té èxit i es retorna un **codi d'error** determinat pel programador al nivell superior.
  - **Opció 3.** El procediment té èxit i s'insereix l'error en un **taula d'errors**.
  - **Opció 4.** El procediment té èxit i l'error es tractat dins del procediment.

### Tipus d'errors que es poden produir:

- Errors predefinits pel propi SGBD, p.e. el codi d'error número 23505 a PostgreSQL vol dir "Unique violation". Es produeixen quan alguna instrucció que s'executa provoca alguna excepció pròpia del SGBD.
- Errors d'usuari. Específics del procediment (P0001 a PostgreSQL). Codi d'error quan dins d'un procediment s'executa una instrucció RAISE EXCEPTION.

Instruccions per gestionar els errors:

- **EXCEPTION:** Accions a dur a terme en cas de que es produeixin excepcions.
- **RAISE EXCEPTION:** Serveix per a que el programador pugui generar els seus propis errors dins d'un procediment

## Gestió d'errors

- Es pot utilitzar el bloc BEGIN... amb la clàusula EXCEPTION per tractar els errors que es produeixen dins d'un procediment.

```
BEGIN  
    statements  
EXCEPTION  
    WHEN condition [ OR condition ... ] THEN  
        handler_statements  
    [ WHEN condition [ OR condition ... ] THEN  
        handler_statements ... ]  
    END;
```

- Si dins del bloc EXCEPTION es produeix una excepció, l'execució del procediment finalitza i l'error es reportat a l'usuari o programa que ha demanat l'execució del procediment emmagatzemat.

### Gestió d'errors: Opció 1 - Captura i retorn d'excepcions

```
CREATE FUNCTION nova_linia_op1(item integer, com integer, qtt integer)  
RETURNS void AS $$  
BEGIN  
    IF (qtt < 12)  
        THEN RAISE EXCEPTION 'Quantitat % inferior a 12', qtt;  
    ELSEIF (qtt > 600)  
        THEN RAISE EXCEPTION 'Quantitat % superior a 600', qtt;  
    END IF;  
    INSERT INTO items_comanda  
        VALUES (item, com, qtt);  
    RETURN;  
  
EXCEPTION  
    WHEN raise_exception THEN  
        RAISE EXCEPTION '%', SQLERRM;  
    WHEN foreign_keyViolation THEN  
        RAISE EXCEPTION 'La comanda o el item no existeixen';  
    WHEN OTHERS THEN  
        RAISE EXCEPTION 'Error intern';  
END;  
$$LANGUAGE plpgsql;
```

## Gestió d'errors: Opció 2 - Captura i retorn d'un codi de retorn

```
CREATE TYPE TError AS (
    codi varchar(5),
    motiu varchar(50));

CREATE FUNCTION nova_linia_op2(item integer, com integer, qtt integer)
RETURNS TError AS $$
DECLARE error TError;
BEGIN
    IF (qtt < 12) THEN RAISE EXCEPTION 'Quantitat % inferior a 12', qtt;
    ELSEIF (qtt > 600) THEN RAISE EXCEPTION 'Quantitat % superior a 600', qtt;
    END IF;
    INSERT INTO items_comanda VALUES (item, com, qtt);
    error.codi:='0';
    RETURN error;
EXCEPTION
    WHEN raise_exception THEN
        error.codi:=SQLSTATE; error.motiu :=SQLERRM;
        RETURN error;
    WHEN foreign_keyViolation THEN
        error.codi:=SQLSTATE; error.motiu:='La comanda o el item no existeixen';
        RETURN error;
    WHEN OTHERS THEN
        error.codi:=SQLSTATE; error.motiu:='Error intern';
        RETURN error;
END;
$$LANGUAGE plpgsql;
```

## Gestió d'errors: Opció 4 – Captura i resolució del motiu de l'error en el mateix procediment

```
CREATE FUNCTION insercions(d1 integer, d2 integer) returns void AS $$

BEGIN
    INSERT INTO prova VALUES (d1,d2);
EXCEPTION
    WHEN undefined_table THEN
        CREATE TABLE prova(
            a integer primary key,
            b integer);

END;
$$LANGUAGE plpgsql;
```

- El procediment intenta recuperar-se de l'error que s'ha produït.
- En aquest exemple, si la taula prova no existeix, es produeix un error. El procediment intenta solucionar-lo creant la taula.
- ALERTA: Una solució d'aquest tipus no sempre és possible !!

## Gestió d'errors: Opció 3 - Captura i inserció a taula d'errors

```
CREATE TABLE t_errors (
    codi varchar(5),
    motiu varchar(50));

CREATE FUNCTION nova_linia_op3(item integer, com integer, qtt integer)
RETURNS void AS $$
BEGIN
    IF (qtt < 12) THEN RAISE EXCEPTION 'Quantitat % inferior a 12', qtt;
    ELSEIF (qtt > 600) THEN RAISE EXCEPTION 'Quantitat % superior a 600', qtt;
    END IF;
    INSERT INTO items_comanda VALUES (item, com, qtt);
    RETURN;
EXCEPTION
    WHEN raise_exception THEN
        INSERT INTO t_errors VALUES (SQLSTATE,SQLERRM);
        RETURN;
    WHEN foreign_keyViolation THEN
        INSERT INTO t_errors VALUES (SQLSTATE,'La comanda o el item no existeixen');
        RETURN;
    WHEN OTHERS THEN
        INSERT INTO t_errors VALUES (SQLSTATE,'Error intern');
        RETURN;
END;
$$LANGUAGE plpgsql;
```

## Sintaxis

```
CREATE TRIGGER nom { BEFORE | AFTER }
{ esdeveniment [ OR ... ] } ON taula
[ FOR [ EACH ] { ROW | STATEMENT } ]
EXECUTE PROCEDURE nomFunc
```

esdeveniment ::= [INSERT | DELETE | UPDATE [of column,.. ]]

Procediment emmagatzemat, específic per a triggers. Agrupa les accions que s'han de fer quan es produeix l'esdeveniment/s.

Begin work

...  
delete from empleats ...

...  
commit

CREATE TRIGGER Esborrar1  
BEFORE delete ON empleats  
FOR EACH STATEMENT  
Execute procedure.....

CREATE TRIGGER Esborrar2  
AFTER delete ON empleats  
FOR EACH STATEMENT  
Execute procedure.....

## Ordre d'execució i visibilitat

### 1. BEFORE STATEMENT :

L'acció s'executa **1 sola vegada abans** de l'execució de la sentència que dispara el disparador.

### 2. BEFORE ROW

L'acció s'executa **1 vegada per a cada tupla** afectada i just abans que la tupla s'insereixi, modifiqui o esborri.

### 3. AFTER ROW

L'acció s'executa **1 vegada per a cada tupla** afectada i després de l'execució de la sentència que dispara el disparador.

### 4. AFTER STATEMENT

L'acció s'executa **1 sola vegada després** de l'execució de la sentència que dispara el disparador.

## Variables accessibles des dels procediments

- **TG\_OP**: conté l'esdeveniment que llança el disparador. Els seus valors poden ser (en majúscules!): INSERT, DELETE i UPDATE.

### ■ OLD, NEW:

- Només tenen valor per triggers FOR EACH ROW.
- Tenen valor NULL en triggers FOR EACH STATEMENT.
- **NEW**: Valors de la tupla després de l'execució de la sentència update o insert. No té valor definit en sentències delete.
- **OLD**: Valors de la tupla abans de l'execució de la sentència update o delete. No té valor definit en sentències insert.

**Exemple:** En un procediments que s'executa degut a un trigger FOR EACH ROW activat per la sentència:

UPDATE items SET qtt=qtt+10 WHERE item=1;

Els valors de les variables NEW i OLD dins del procediment seran:

OLD.item = 1  
OLD.qtt = 100  
NEW.item = 1  
NEW.qtt = 110

## Sintaxis dels procediments específics per a disparadors

```
CREATE FUNCTION disp_procediment()
RETURNS trigger AS $$
BEGIN
...
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Procediment que no rep cap paràmetre i retorna un tipus especial de postgres anomenat trigger

Només pot retornar: NULL, NEW o OLD.  
(veure transparència següent)

## Return dels procediments BEFORE, FOR EACH ROW

Aquests procediments poden retornar els valors següents:

- **NEW**, per indicar que l'execució del procediment per la fila actual ha d'acabar normalment i que l'operació que dispara el disparador (INSERT/UPDATE) s'ha d'executar. En aquest cas, el procediment pot retornar el valor original de la variable NEW o modificar-ne el contingut. Si el procediment modifica el contingut de la variable NEW, està modificant directament la fila que s'inserirà o modificarà.
- **OLD**, per indicar que l'execució del procediment per la fila actual ha d'acabar normalment i que l'operació que dispara el disparador (DELETE/UPDATE) s'ha d'executar. En el cas de l'UPDATE si el procediment retorna OLD el UPDATE no fa la modificació de la fila actual.
- **NULL**, per indicar que l'execució del procediment per la fila actual ha d'acabar normalment i que l'operació que dispara el disparador (INSERT/ UPDATE/ DELETE) no s'arriba a executar.

## Exemple: Return dels procediments BEFORE, FOR EACH ROW, INSERT

```
CREATE TABLE t(
    a integer PRIMARY KEY,
    b integer);

CREATE FUNCTION prog()
.....
.....
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trig_i BEFORE INSERT ON t FOR EACH ROW
EXECUTE PROCEDURE prog();

INSERT INTO t VALUES (1,2);
```

El procediment prog() pot retornar dos valors, segons el return usat:

- RETURN NULL. En aquest cas, la fila <1,2> no s'insereix a la taula t.
- RETURN NEW. En aquest cas, tenim dues possibilitats. Si el valor de la variable NEW no ha sigut modificat pel procediment prog(), aleshores s'executa l'inserció de la a fila <1,2> a la taula t. Si el valor de la variable NEW ha sigut modificat pel procediment prog(), per exemple s'ha executat l'operació NEW.b=3, aleshores s'executa l'inserció de la fila <1,3> a la taula t.

## Exemple: Return dels procediments BEFORE, FOR EACH ROW, UPDATE

```
CREATE TABLE t(
    a integer PRIMARY KEY,
    b integer);

CREATE FUNCTION prog()
.....
.....
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trig_u BEFORE UPDATE ON t FOR EACH ROW
EXECUTE PROCEDURE prog();

UPDATE t SET b=3 WHERE a=1;
```

El procediment prog() pot retornar tres valors, segons el return usat:

- RETURN NULL o RETURN OLD. En aquest cas, la fila a=1 no es modifica en la taula t.
- RETURN NEW. En aquest cas, tenim dues possibilitats. Si el valor de la variable NEW no ha sigut modificat pel procediment prog(), aleshores s'executa la modificació de la fila a=1 passant a ser <1,3>. Si el valor de la variable NEW ha sigut modificat pel procediment prog(), per exemple s'ha executat l'operació NEW.b=5, aleshores s'executa la modificació de la fila a=1 passant a ser <1,5>.

## Exemple: Return dels procediments BEFORE, FOR EACH ROW, DELETE

```
CREATE TABLE t(
    a integer PRIMARY KEY,
    b integer);

CREATE FUNCTION prog()
.....
.....
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trig_d BEFORE DELETE ON t FOR EACH ROW
EXECUTE PROCEDURE prog();

DELETE FROM t WHERE a=1;
```

El procediment prog() pot retornar tres valors, segons el return usat:

- RETURN NULL. En aquest cas, no s'executa l'esborrat de la fila a=1 la taula t.
- RETURN OLD. En aquest cas, s'executa l'esborrat de la fila a=1 en la taula t.

## Valors de les variables NEW i OLD: RESUM

	ROW			STATEMENT
	INSERT	UPDATE	DELETE	
BEFORE	<ul style="list-style-type: none"><li>■ OLD: No té valor definit.</li><li>■ NEW: Valors de la tupla després de l'insert</li></ul>	<ul style="list-style-type: none"><li>■ OLD: Valors de la tupla abans de l'update.</li><li>■ NEW: Valors de la tupla després de l'update</li></ul>	<ul style="list-style-type: none"><li>■ OLD: Valors de la tupla abans del delete.</li><li>■ NEW: No té valor definit.</li></ul>	Valor NULL
AFTER				

ATENCIO: La modificació del valor de la variable NEW dins d'un procediment d'un trigger BEFORE INSERT / UPDATE pot afectar a l'efecte de la sentència que ha activat el trigger.

## Return dels procediments: RESUM

	ROW	STATEMENT
BEFORE	<ul style="list-style-type: none"> <li>■ RETURN NEW           <ul style="list-style-type: none"> <li>- A continuació s'executa la sentència que activa el disparador en triggers INSERT, UPDATE. I si es modifica el valor de la variable NEW dins del procediment, s'executa la sentència amb el valor modificat.</li> <li>- No fa res en triggers DELETE</li> </ul> </li> <li>■ RETURN OLD           <ul style="list-style-type: none"> <li>- A continuació s'executa la sentència que activa el disparador en triggers DELETE, UPDATE. Si és un trigger UPDATE, no fa el canvi establert en la sentència.</li> <li>- No es fa res en triggers INSERT.</li> </ul> </li> <li>■ RETURN NULL           <ul style="list-style-type: none"> <li>- No executa la sentència que activa el disparador</li> </ul> </li> </ul>	Valor de retorn ignorat. Millor: RETURN NULL
AFTER	Valor de retorn ignorat. Millor: RETURN NULL	Valor de retorn ignorat. Millor: RETURN NULL

## Exemple 1: Auditoria

- Objectiu: Mantenir un registre de les modificacions que fan els usuaris a la taula items. Cada vegada que es modifiqui la quantitat d'un item haurem de guardar un registre a la taula log\_record amb: l'identificador de l'item, l'usuari que ha fet la modificació, la data en que s'ha fet, i la quantitat inicial i quantitat final d'item.

```

CREATE TABLE log_record(
    item integer,
    username char(8),
    update_time timestamp,
    old_qtt integer,
    new_qtt integer);

create table items(
    item integer primary key,
    name char(25),
    qtt integer,
    preu_total decimal(9,2));

CREATE FUNCTION insert_log() RETURNS trigger AS $$
BEGIN
    insert into log_record values (OLD.item,current_user,current_date,OLD.qtt,NEW.qtt);
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER restrict_after_empl
AFTER UPDATE of qtt ON items
FOR EACH ROW EXECUTE PROCEDURE insert_log();
  
```

## Exemple 2: Auditoria

Objectiu: Mantenir un registre de les modificacions que fan els usuaris a la taula items. Ara, només voldrem guardar l'usuari que fa la modificació i la data en que es produeix la modificació.

```

CREATE TABLE log_record2(
    username char(8),
    update_time timestamp);

CREATE FUNCTION inserta_log() RETURNS trigger AS $$ 
BEGIN
    insert into log_record2 values (current_user,current_date);
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER audit_items
AFTER UPDATE ON items
FOR EACH STATEMENT EXECUTE PROCEDURE inserta_log();
  
```

## Exemple 3: Atribut derivat

- Objectiu: Donada la taula items mantenir de manera automàtica l'atribut derivat preu\_total quan hi ha modificacions de la quantitat d'estoc.

```

CREATE FUNCTION calcular_nou_total() RETURNS trigger AS $$
BEGIN
    IF (old.qtt<>0) THEN
        NEW.preu_total:=((OLD.preu_total/OLD.qtt)*NEW.qtt);
    END IF;
    RETURN NEW;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER atribut_derivat
BEFORE UPDATE OF qtt ON items
FOR EACH ROW EXECUTE PROCEDURE calcular_nou_total();
  
```

Modificació de la variable NEW dins del procediment.

El procediment fa que, a part de modificar-se la quantitat en estoc d'un item, es modifiqui al mateix temps el preu\_total de l'item.

## Exemple 4 : Regla de negoci

- Objectiu: Una única sentència de modificació no pot augmentar la quantitat total en estoc dels productes en més d'un 50%.

```
CREATE TABLE TEMP(old_qtt integer);

CREATE FUNCTION update_items_before() RETURNS trigger AS $$
BEGIN
    INSERT INTO temp SELECT sum(qtt) FROM items;
    return null;
END $$ LANGUAGE plpgsql;

CREATE FUNCTION update_items_after() RETURNS trigger AS $$
DECLARE
    oldqtt integer default 0;
    newqtt integer default 0;
BEGIN
    SELECT old_qtt into oldqtt FROM temp;
    DELETE FROM temp;
    SELECT sum(qtt) into newqtt FROM items;
    IF (newqtt>oldqtt*1.5) THEN RAISE EXCEPTION 'Violació regla de negoci';
    END IF;
    RETURN NULL;
END $$ LANGUAGE plpgsql;

CREATE TRIGGER regla_negociBS BEFORE UPDATE ON items
    FOR EACH STATEMENT EXECUTE PROCEDURE update_items_before();

CREATE TRIGGER regla_negociAS AFTER UPDATE ON items
    FOR EACH STATEMENT EXECUTE PROCEDURE update_items_after();
```

## Exemple 4: Regla de negoci – Solució Incremental

```
CREATE TRIGGER regla_negociBS
    BEFORE UPDATE ON items
    FOR EACH STATEMENT
    EXECUTE PROCEDURE update_items_before();

CREATE TRIGGER regla_negociBR
    BEFORE UPDATE OF qtt ON items
    FOR EACH ROW
    EXECUTE PROCEDURE update_items_inc();
```

Cal notar que el segon disparador en la solució incremental és **BEFORE, FOR EACH ROW**. Amb aquesta solució s'estalvia l'accés a tots els items per a calcular l'estoc després de l'update, ja que el procediment **update\_items\_inc** utilitz a la informació sobre les tuples modificades que hi ha a les variables NEW i OLD abans de cada update.

## Exemple 4: Regla de negoci – Solució Incremental

- Objectiu: No pot ser que una única sentència de modificació augmenti la quantitat total en estoc dels productes en més d'un 50%

```
CREATE TABLE TEMP(
    old_qtt integer,
    incr    integer);

CREATE FUNCTION update_items_before() RETURNS trigger AS $$
BEGIN
    DELETE From temp;
    INSERT INTO temp(old_qtt,incr) select sum(qtt),0 from items;
    return null;
END $$ LANGUAGE plpgsql;

CREATE FUNCTION update_items_inc() RETURNS trigger AS $$
DECLARE
    oldqtt integer default 0;
    suma_incr integer default 0;
BEGIN
    UPDATE temp
    SET incr=incr+(NEW.qtt-OLD.qtt);
    SELECT old_qtt,incr INTO oldqtt,suma_incr FROM temp;
    IF (suma_incr>oldqtt*0.5) THEN
        RAISE EXCEPTION 'Violació regla de negoci';
    END IF;
    return NEW;
END $$ LANGUAGE plpgsql;
```

A diferència de l'anterior solució en aquesta no s'accedeix a totes les files de la taula *items*, sinó només a les que s'ha de modificar.  
Per aquest motiu es diu que aquesta és una solució incremental.

## Exemple 5: Auditoria

- Objectiu: Auditar els esborrats i modificacions de la taula *items*. Cada vegada que s'executi una sentència d'esborrat o modificació de files de la taula, cal registrar a la taula *log\_record2* el nom de l'usuari que ha invocat la sentència, l'instant en què s'ha produït, i l'operació concreta que s'ha executat (delete o update).

```
CREATE TABLE log_record2(
    username char(8),
    update_time timestamp,
    operacio varchar(6));

CREATE FUNCTION inserta_log() RETURNS trigger AS $$
BEGIN
    insert into log_record2
        values (current_user,current_date,TG_OP);
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER audit_items
AFTER UPDATE OR DELETE ON items
FOR EACH STATEMENT EXECUTE PROCEDURE inserta_log();
```

Veure  
transparència  
Variables  
accessibles des  
dels procediments

## Exemple 6 – Manteniment de restriccions

- Objectiu: Tot estudiant ha de ser usuari.

```
CREATE TABLE usuari(
    id_u integer primary key);
CREATE TABLE estudiant(
    id_e integer references usuari);

CREATE FUNCTION inserir() RETURNS trigger AS $$ 
BEGIN
    IF (NOT EXISTS (SELECT * FROM usuari WHERE id_u=NEW.id_e))
    THEN
        INSERT INTO usuari VALUES (NEW.id_e);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER restrict1 BEFORE INSERT ON estudiant
FOR EACH ROW EXECUTE PROCEDURE inserir();
```

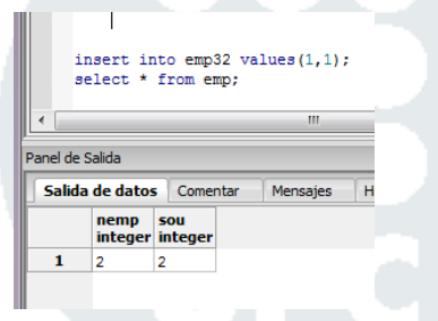
**Precedència de restriccions:** Tenint en compte la precedència entre disparadors i restriccions, penseu què passaria si el disparador de l'exemple s'hagués definit AFTER, en lloc de BEFORE.

## INSTEAD OF: un altre tipus de trigger per actualitzar vistes

```
create table emp(nemp integer primary key, sou integer);
create view emp32 as select * from emp where nemp>32;
```

```
CREATE or replace FUNCTION insert32() RETURNS trigger AS $$ 
BEGIN
    insert into emp values (new.nemp+1,new.sou+1);
    RETURN new;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER view_insert
  INSTEAD OF INSERT ON emp32
  FOR EACH ROW
  EXECUTE PROCEDURE insert32();
```



## Interferències entre transaccions

- Les interferències es produeixen si les transaccions no s'aïllen adequadament entre si. Distingim quatre grans tipus d'interferències:
  - Actualització perduda
  - Lectura no confirmada
  - Lectura no repetible
  - Anàlisi inconsistent i fantasmes
- Actualització perduda:** Aquesta interferència es produeix quan es perd el canvi que ha efectuat una operació d'escriptura:

Transacció T <sub>1</sub> (reintegrat de 10)	Transacció T <sub>2</sub> (reintegrat de 25)
Saldo := llegir saldo(Compte)	Saldo := llegir saldo(Compte)
escriure saldo(Compte, Saldo - 10)	escriure saldo(Compte, Saldo - 25)
COMMIT	COMMIT

## Interferències entre transaccions

- Lectura no confirmada:** Aquesta interferència es produeix quan una transacció T1 llegeix una dada, que ha estat modificada per una altra transacció T2, i que és una dada que no és definitiva (per exemple: després la transacció T2 avorta, o després la transacció T2 la torna a modificar)

Transacció T <sub>1</sub> (reintegrat de 10)	Transacció T <sub>2</sub> (reintegrat de 25)
	Saldo := llegir saldo(Compte)
	escriure saldo(Compte, Saldo - 25)
Saldo := llegir saldo(Compte)	

## Interferències entre transaccions

- Lectura no repetible:** Aquesta interferència es produeix si una transacció llegeix dues vegades la mateixa dada i obté valors diferents, a causa d'una modificació efectuada per una altra transacció

Transacció T <sub>1</sub> (lectura de saldo)	Transacció T <sub>2</sub> (reintegratament de 25)
Saldo := llegir saldo(Compte)	
	Saldo := llegir saldo(Compte)
	escriure saldo(Compte, Saldo - 25)
	COMMIT
Saldo := llegir saldo(Compte)	
COMMIT	

## Interferències entre transaccions

- Anàlisi inconsistent:** Els tres tipus d'interferències anteriors tenen lloc respecte a una única dada de la BD, però també es produeixen interferències respecte a la visió que dues transaccions tenen d'un conjunt de dades.

Un exemple de situació en què passa:

Transacció T1 (consulta de saldo)	Transacció T2 (transferència)
saldo2:= llegir saldo(compte2)	
	saldo1:= llegir saldo(compte1)
	escriure saldo(compte1, saldo1-100)
saldo1:= llegir saldo(compte1)	
COMMIT	
	saldo2:= llegir saldo(compte2)
	escriure saldo(compte2, saldo2+100)
	COMMIT

## Interferències entre transaccions

**Fantasmes (Cas particular d'Anàlisi Inconsistent).** Aquesta interferència es pot produir quan una transacció llegeix un conjunt de dades relacionat i existeix una altra transacció que afegeix noves dades a aquest conjunt. Suposem el següent:

- T<sub>1</sub> llegeix tots els registres que verifiquen una condició C
- T<sub>2</sub> actualitza (insereix o modifica) un registre, que no complia la condició C, de manera que passa a complir-la
- T<sub>1</sub> torna a llegir els registres inicials o alguna informació que en depèn

Transacció T <sub>1</sub> (llistat de comptes)	Transacció T <sub>2</sub> (creació de comptes)
Llegir tots els comptes del banc. Obtenim Compte1 i Compte2	crear_compte(Compte3)
	escriure saldo(Compte3, 100)
Sumar els saldo de tots els comptes. Obtenim saldo de Compte1 + + saldo de Compte2 + 100  (El Compte3, amb saldo 100, és el fantasma)	
COMMIT	COMMIT

## Interferències entre transaccions

**Fantasmes :** compte1 i compte2 són de clients de Barcelona, compte4 és d'un client de Tarragona.

Transacció T1 (llistat de comptes clients de Barcelona)	Transacció T2 (canvi residència)
llegir comptes clients Barcelona	
mostrar dades compte1	
mostrar dades compte2	
	canviar_residència(compte4, Barcelona)
sumar el saldo de tots els comptes de clients de Barcelona	
obtenim saldo compte1+ saldo compte2+ saldo compte4	
(el compte4 és el fantasma)	
COMMIT	
	COMMIT

## Serialitzabilitat

- La **teoria de la serialitzabilitat** defineix de manera precisa les condicions que s'han de complir per a considerar que les transaccions estan aïllades entre si correctament

*Important: la teoria de la serialitzabilitat assumeix que les transaccions sempre confirmen els seus resultats*

- Conceptes:
  - Grànul
  - Horari o història
  - Horari serial
  - Accions conflictives o no commutables
  - Graf de precedències
  - Horari serialitzable

- Accions de lectura ( $R(G)$ ,  $RU(G)$ ) i accions d'escriptura ( $W(G)$ )
- $G$  és el grànul, és a dir, la unitat de dades controlada individualment per l'SGBD (pot variar en funció del gestor): pàgina (bloc) de disc, un registre (una tupla), etc.



## Traducció de sentències SQL a operacions sobre grànuls:

### Sentència SQL

SELECT

### Operació sobre grànuls

$R(G)$   
R: Lectura del granul G

INSERT/UPDATE/DELETE

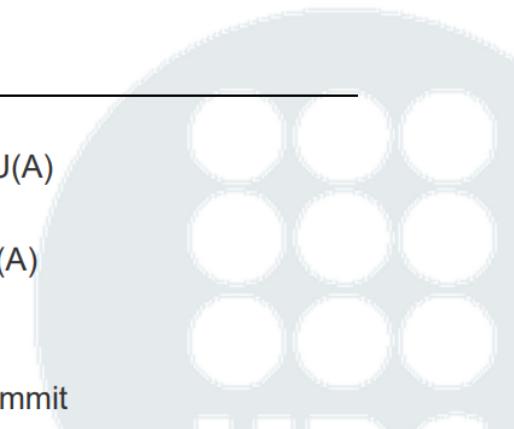
$RU(G) W(G)$   
RU: Lectura amb intenció de modificació posterior del granul G.  
Indica que la transacció voldrà executarà més endavant  $W(G)$ .  
 $W$ : Escritura del granul G

Les sentències SQL poden donar lloc a una o més operacions sobre granuls. Per exemple, en el cas d'un select que accedeix a tres files, i que el granul sigui la fila, la traducció donaria lloc a tres reads, un per cada fila.

## Serialitzabilitat: Horari o història

- L'execució concurrent d'un conjunt de transaccions (a on es preserva l'ordre d'accions dins de cada transacció) rep el nom **d'horari o història**:

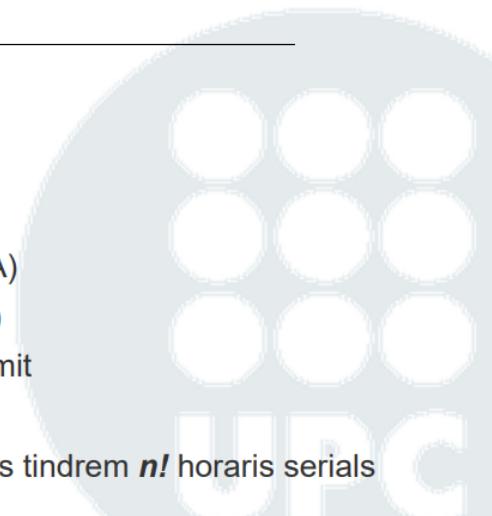
# temps	$T_1$	$T_2$
1	$R(A)$	
2		$RU(A)$
3	$R(B)$	
4		$W(A)$
5	$R(C)$	
6	commit	
7		commit



## Serialitzabilitat: Horari serial

- Un **horari serial** és aquell a on no hi ha encavalcament entre les accions de les transaccions implicades:

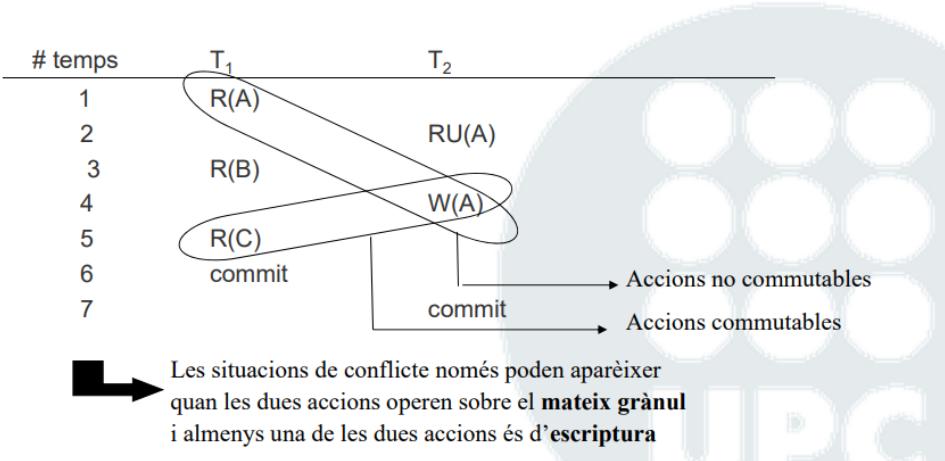
# temps	$T_1$	$T_2$
1	$R(A)$	
2	$R(B)$	
3	$R(C)$	
4	commit	
5		$RU(A)$
6		$W(A)$
7		commit



Donat un conjunt de  $n$  transaccions tindrem  $n!$  horaris serials possibles

## Serialitzabilitat: Accions conflictives

- En un horari, dues accions es consideren **accions conflictives** (o **accions no commutables**) si pertanyen a transaccions distinta i l'ordre en què s'executen pot afectar el valor del grànul que hagi llegit una de les transaccions o el valor final del grànul



## Serialitzabilitat: Horari serializable

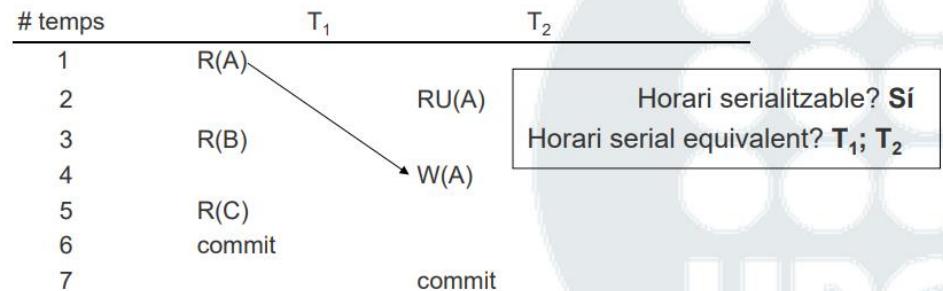
- Un horari (història) es considera correcte si l'ordre relatiu de tots els parells d'accions no commutables és el mateix que en algun horari serial. En aquest cas, el graf de precedències no té cicles



Els horaris correctes s'anomenen **horaris serialitzables**

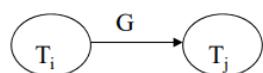


Un horari serializable **sempre** produeix **el mateix resultat** que algun horari serial

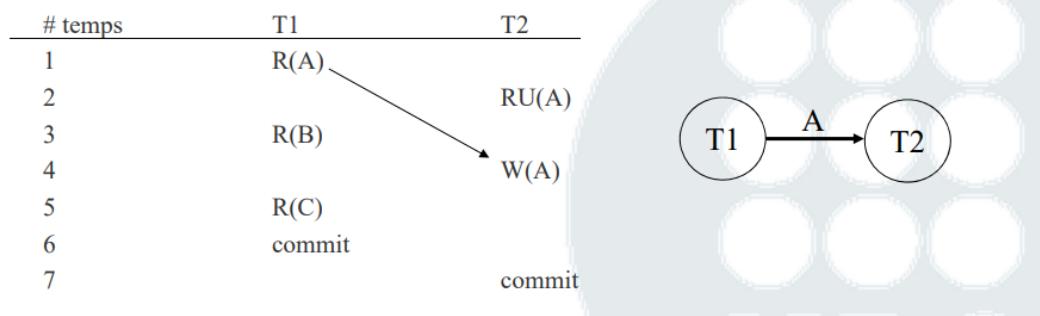


## Serialitzabilitat: Horari no serializable

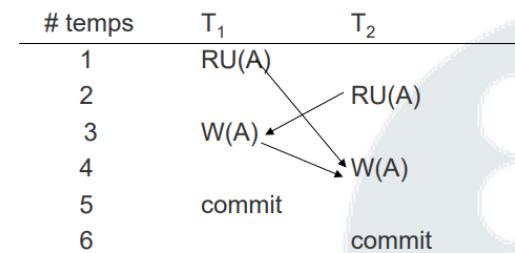
### Serialitzabilitat: Graf de precedències



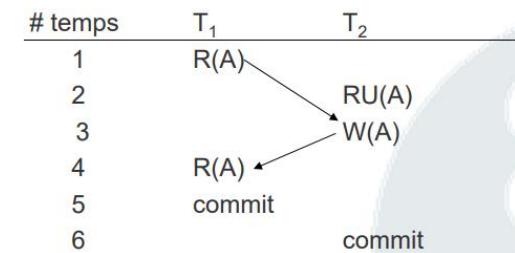
Si  $\exists op_i(G) \in T_i \wedge \exists op_j(G) \in T_j$  tal que:  
 -  $op_i \wedge op_j$  no són commutables  
 - i  $op_i$  s'ha executat abans que  $op_j$



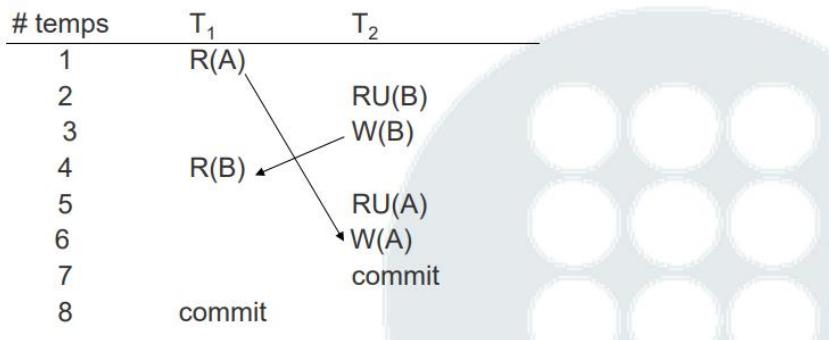
### Actualització Perduda



### Lectura no repetible



## Anàlisi inconsistent



## Serialitzabilitat: Horari no serializable

### Fantasma

# temps	T <sub>1</sub>	T <sub>2</sub>
1	R(IC)	
2		R(C1) R(C2)
3		
4		RU(C3)
5		W(C3)
6		RU(IC)
7		W(IC)
8	R(IC)	
9	R(C1)	
10	R(C2)	
11	R(C3)	
12	commit	
13		commit

## Tècniques per al control de concorrència

- Un SGBD pot resoldre les interferències entre transaccions de dues maneres possibles:
  - Cancel·lar automàticament transaccions problemàtiques i desfer-ne els canvis
  - Suspender'n l'execució temporalment i reprendre-la quan desaparegui el perill d'interferència.

↓

*disminució en el nivell de paral·lelisme o nivell de concorrència del sistema*
- El **nivell de paralelisme** és la feina efectiva realitzada, és a dir, la feina realment útil per als usuaris, efectuada per unitat de temps

## Recuperabilitat

- Hem vist que algunes interferències es produeixen quan cancel·lem transaccions. Cancel·lar una transacció suposa desfer-ne tots els canvis i recuperar el valor anterior dels grànuls
- La serialitzabilitat ignora la possibilitat de cancel·lacions. Per tant, per evitar les interferències que provoquen hem d'exigir noves condicions a l'execució de les transaccions
- Un horari compleix el **criteri de recuperabilitat** si cap transacció T<sub>2</sub> que llegeix o escriu un grànul escrit per una altra transacció T<sub>1</sub> confirma sense que abans ho hagi fet T<sub>1</sub>

# temps	T <sub>1</sub>	T <sub>2</sub>
1	RU(A)	
2		W(A)
3		
4		
5		RU(A) W(A) commit
6	abort	

## Control de concorrència amb reserves

- Els SGBD garanteixen l'aïllament gràcies a les tècniques de control de concorrència.
- La tècnica més utilitzada per dur a terme el control de concorrència és la tècnica de reserves.
- La idea bàsica de l'ús de reserves és que una transacció ha d'obtenir una **reserva d'un grànul amb una certa modalitat** abans de poder efectuar accions (lectures o escriptures) sobre el grànul.
- Una transacció T pot demanar una reserva sobre un grànul G si executa l'acció LOCK(G,m) a on m és una modalitat que permet executar les accions desitjades sobre G.
- Distingirem dues modalitats:
  - Modalitat compartida (S, **Shared**): permet fer lectures
  - Modalitat exclusiva (X, **eXclusive**): permet fer lectures i escriptures
- Per alliberar una reserva d'un grànul G, caldrà que la transacció T executi l'acció UNLOCK(G).

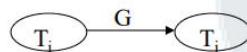
## Control de concorrència amb reserves

- Quan una transacció demana una reserva, l'SGBD decideix si la pot concedir, cosa que farà si el tipus de reserva que se li demana no és incompatible amb cap de les reserves ja atorgades a altres transaccions sobre el mateix grànul

	Compartida (S)	Exclusiva (X)
Compartida (S)	Sí	No
Exclusiva (X)	No	No

- Si una reserva no es pot concedir, se suspèn l'execució de la transacció. Sempre que s'allibera una reserva d'un grànul, l'SGBD mira si pot reprendre l'execució d'alguna transacció que s'hagi suspès a causa d'aquell grànul

### Graf d'espera



Si  $T_i$  està esperant a adquirir una reserva sobre un grànul  $G$  que ha estat concedida a  $T_j$

## Control de concorrència amb reserves

# temps	T <sub>1</sub>	T <sub>2</sub>
1	LOCK(A, S)	
2	R(A)	
3		LOCK(A, X)
4	LOCK(B, S)	.
5	R(B)	.
6	LOCK(C, S)	.
7	R(C)	
8	UNLOCK(A)	
9		RU(A)
10		W(A)
11	UNLOCK(B)	
12	UNLOCK(C)	
13		UNLOCK(A)
14	commit	
15		commit

## Control de concorrència amb reserves

- Tot i que els serveis de petició i alliberament de reserves són la base sobre la qual es construeix el control de concorrència, per si mateixos no garanteixen l'aïllament de les transaccions:

# temps	T <sub>1</sub>	T <sub>2</sub>
1	LOCK(A, S)	
2	R(A)	
3	UNLOCK(A)	
4		LOCK(A, X)
5		RU(A)
6		W(A)
7		UNLOCK(A)
8	LOCK(A, S)	
9	R(A)	
10	UNLOCK(A)	
11	commit	
12		commit

La interferència (lectura no repetible) es produeix igualment!!!

### Control de concorrència amb reserves: PR2F

- Per aconseguir l'aïllament de les transaccions, les transaccions han de seguir unes certes normes a l'hora de demanar i alliberar reserves
- Una transacció segueix el **protocol de reserves en dues fases** (PR2F) si reserva qualsevol grànul en la modalitat adequada abans d'operar-hi, i mai no adquireix una reserva de qualsevol grànul després d'haver-ne alliberat qualsevol altre abans.
- Teorema:** Si les transaccions segueixen el PR2F i fan commit → Horari serialitzable

# temps	T <sub>1</sub>	T <sub>2</sub>
1	LOCK(A, S)	
2	R(A)	
3		LOCK(A, X)
4	R(A)	.
5	UNLOCK(A)	.
6		
7		commit
8		
9		RU(A)
10		W(A)
		UNLOCK(A)
		commit

Horari sense interferències (horari serializable) amb horari serial equivalent  $T_1; T_2$

## Control de concorrència amb reserves: PR2F (fortament) estricta

- Parlem ara de recuperabilitat

# temps	T <sub>1</sub>	T <sub>2</sub>
1	LOCK(A,X)	
2	RU(A)	
3	W(A)	
4	UNLOCK(A)	
5		LOCK(A,S)
6		R(A)
7		UNLOCK(A)
8		commit
9	abort	

Malgrat que T<sub>1</sub> i T<sub>2</sub> segueixen el PR2F la interferència (lectura no confirmada) es produeix igualment si deixem que T<sub>2</sub> confirmi els seus resultats  $\Rightarrow$  necessitem que les reserves es mantinguin fins l'acabament de les transaccions



**Protocol de reserves en dues fases (fortament) estricta**  
(PR2F+ reserves fins l'acabament de les transaccions)

## Control de concorrència amb reserves: PR2F estricta

# temps	T <sub>1</sub>	T <sub>2</sub>
1	LOCK(A,X)	
2	RU(A)	
3	W(A)	
4	abort(+ UNLOCK(A))	
5		LOCK(A,S)
6		⋮
7		R(A)
		commit(+ UNLOCK(A))

Aplicació del protocol de reserves en dues fases estricta

## Reserves i relaxació del nivell d'aïllament

- Hem assumit que és necessari que l'SGBD garanteixi la serialitzabilitat i la recuperabilitat de les transaccions, proporcionant una protecció total respecte davant de qualsevol tipus d'interferència
- Les principals conseqüències d'aquest fet són:
  - $\Delta$  sobrecàrrega de l'SGBD en termes de gestió d'informació de control
  - $\nabla$  del nivell de paralel·isme
- En determinades circumstàncies és convenient relaxar el nivell d'aïllament i possibilitar que es produixin interferències; això és correcte si se sap que aquestes interferències no es produiran realment o si no és important que es produueixin
- L'SQL estàndard ens permet relaxar el nivell d'aïllament de la manera següent:

```
SET TRANSACTION mode_acces
ISOLATION LEVEL nivell_aïllament
a on:
- Mode d'accés: READ ONLY o bé READ WRITE
- Nivell d'aïllament: READ UNCOMMITTED, READ COMMITTED,
REPEATABLE READ o bé SERIALIZABLE
```

## Reserves i nivells d'aïllament

	Actualització perduda	Lectura no confirmada	Lectura no repetible i anàlisi inconsistent (tret de fantasma)	Fantasma
READ UNCOMMITTED	Sí	No	No	No
READ COMMITTED	Sí	Sí	No	No
REPEATABLE READ	Sí	Sí	Sí	No
SERIALIZABLE	Sí	Sí	Sí	Sí

- READ UNCOMMITTED protegeix les dades actualitzades, evitant que cap altra transacció les actualitzi, fins que acaba la transacció
- READ COMMITTED protegeix parcialment les lectures, impedint que una altra transacció llegeixi dades que encara no s'han confirmat
- REPEATABLE READ impedeix, fins que acaba la transacció, que una altra transacció actualitzi una dada que s'hagi llegit
- SERIALIZABLE ofereix aïllament total i evita qualsevol tipus d'interferència incloent-hi els fantasma

# Reserves i nivells d'aïllament

## Reserves i nivells d'aïllament

### READ UNCOMMITTED

- Reserves X fins l'acabament de la transacció.
- No es fan reserves S per lectura

### READ COMMITTED

- Reserves X fins l'acabament de la transacció.
- Reserves S fins després de la lectura

### REPEATABLE READ

- Reserves X i S fins l'acabament de la transacció

### SERIALIZABLE

- Totes les reserves fins l'acabament de les transaccions
- (incloent reserves d'informació de control)

**READ UNCOMMITTED** Reserves X fins l'acabament de la transacció. No es fan reserves S per lectura

Actualització perduda		Lectura no confirmada	
T1	T2	T1	T2
Lock(A,X) RU(A)		Lock(A,X) RU(A) W(A)	
W(A)			Lock(A,X)
C(Unlock(A))		RU(A) W(A)	
		C(Unlock(A))	
			C
			A(Unlock(A))

L'interferència s'evita. HSE: T1;T2

### Lectura no repetible

T1	T2
R(A)	Lock(A,X) RU(A) W(A)
R(A) C	C(Unlock(A))

L'interferència no s'evita

L'interferència no s'evita

### Fantasmes

T1	T2
R(IC)	R(C1) R(C2)
R(IC)	C
	C(Unlock(A))

L'interferència no s'evita

## Reserves i nivells d'aïllament:

### Actualització perduda

T <sub>1</sub>	T <sub>2</sub>
RU(A)	RU(A)
W(A)	W(A)
commit	commit

### Lectura no confirmada

T <sub>1</sub>	T <sub>2</sub>
RU(A)	RU(A)
W(A)	R(A)
commit	commit

### Fantasmes

T <sub>1</sub>	T <sub>2</sub>
R(IC)	R(C3)
R(C1)	W(C3)
R(C2)	RU(IC)
	W(IC)
	commit

### Lectura no repetible

T <sub>1</sub>	T <sub>2</sub>
R(A)	RU(A)
	W(A)
R(A)	
commit	commit

**READ COMMITTED** Reserves X fins l'acabament de la transacció. Reserves S fins després de la lectura

### Actualització perduda s'evita

Lectura no confirmada	Fantasmes
T1	T2
Lock(A,X) RU(A) W(A)	R(IC) Lock(C1,S) R(C1)
	Unlock(C1) Lock(C2,S) R(C2)
	Unlock(C2)

L'interferència s'evita. HSE: T1;T2

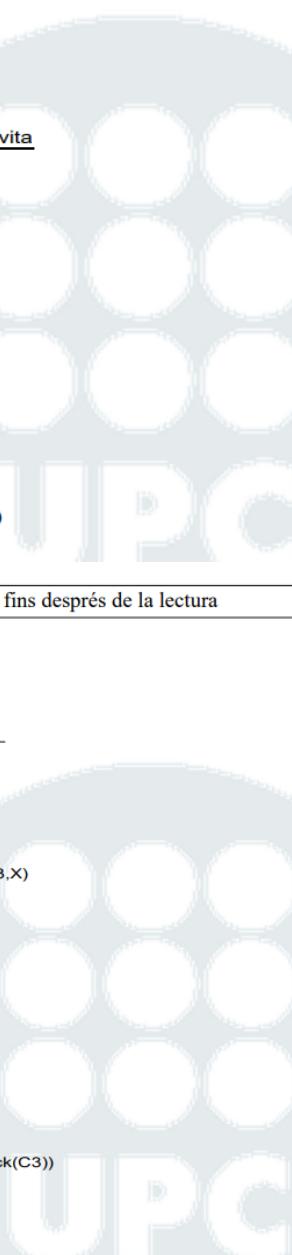
### Lectura no repetible

T1	T2
Lock(A,S) R(A) Unlock(A)	Lock(A,X) RU(A) W(A)
Lock(A,S)	
	C(Unlock(A))

L'interferència no s'evita

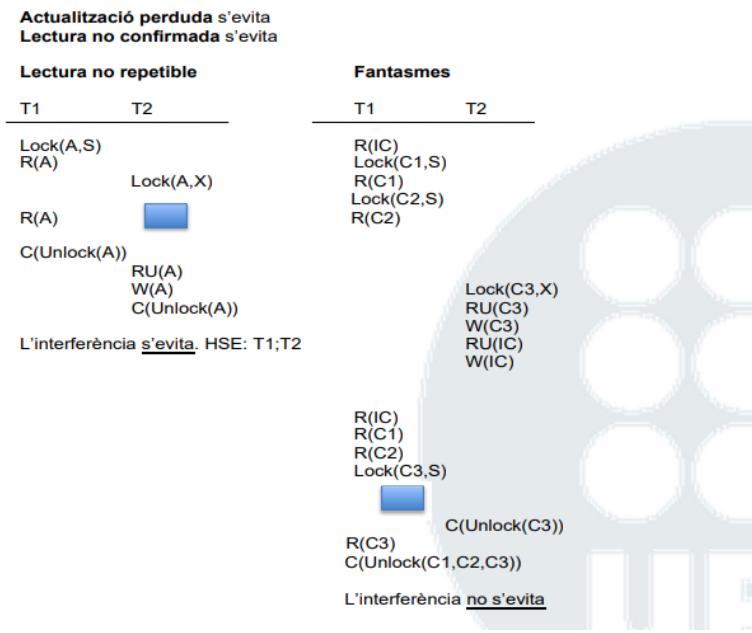
Lectura no confirmada	Fantasmes
T1	T2
Lock(A,S) R(A) Unlock(A)	R(IC) Lock(C1,S) R(C1)
	Unlock(C1) Lock(C2,S) R(C2)
	Unlock(C2)

L'interferència no s'evita

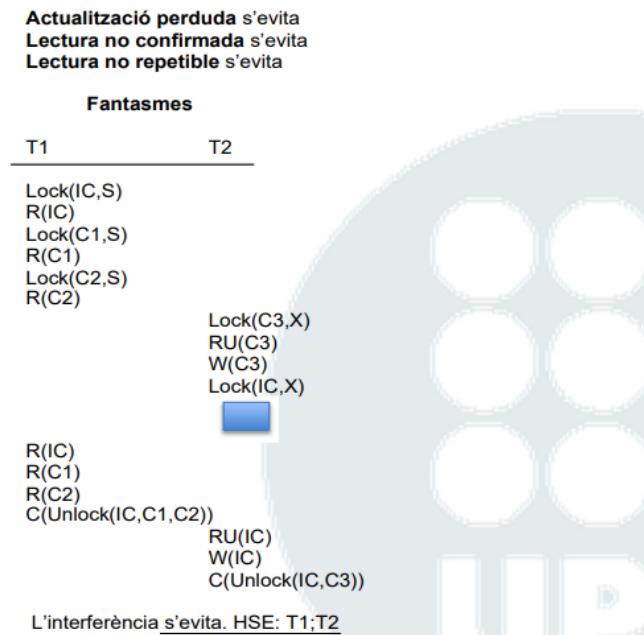


# Reserves i nivells d'aïllament

**REPEATABLE READ** Reserves X i S fins l'acabament de la transacció

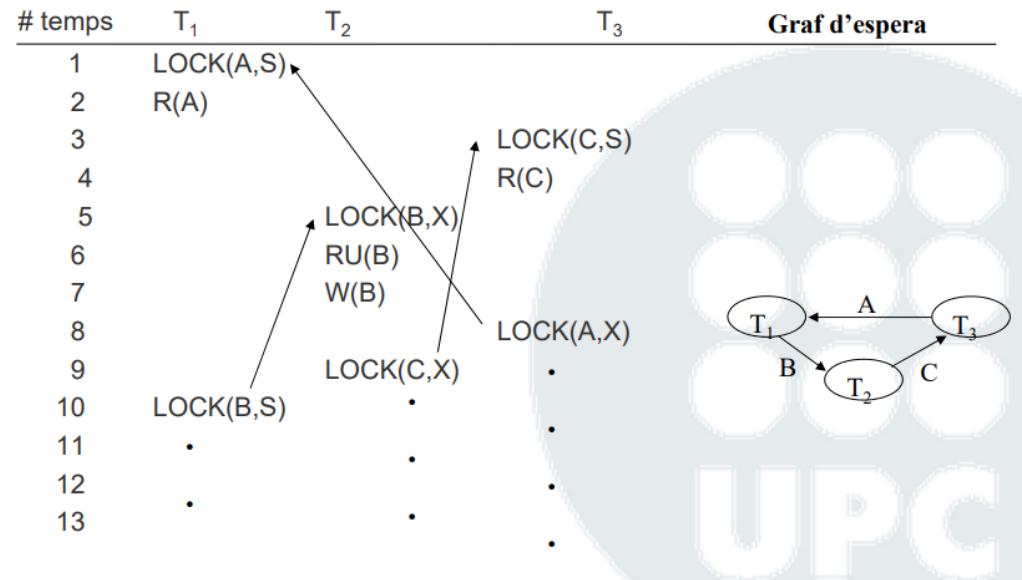


**SERIALIZABLE** Totes les reserves fins l'acabament de les transaccions  
(incloent reserves d'informació de control)



## Control de concorrència amb reserves: Abraçades mortals

- Problema inherent a les tècniques basades en reserves  $\Rightarrow$  possibilitat de que es produueixin **esperes indefinides** (abraçades mortals)



## Control de concorrència amb reserves: Abraçades mortals

- Davant de la possibilitat que es produueixin abraçades mortals, els SGBD basats en reserves han d'optar per una de les tres possibilitats següents:
  - Prevenir-les abans que es produueixin
  - **Detectar-les i resoldre-les una vegada s'hagin produït**
  - Definir un temps d'espera màxim que, un cop superat, faci que es cancel·li automàticament la transacció
- Un SGBD detecta les abraçades mortals buscant cicles en el graf d'espera. Aquesta cerca la pot fer en moments diferents:
  - Sempre que una transacció demana una reserva i no l'obté immediatament
  - Cada cert temps
  - Quan tenim transaccions que triguen massa temps en acabar
- Un cop que l'SGBD detecta l'abraçada mortal, la única cosa que pot fer és trencar el cicle cancel·lant una o diverses de les transaccions implicades

## Recuperació

- Els protocols de recuperació de l'SGBD han de garantir l'atomicitat i la definitivitat de les transaccions.
- Objectius: mai no es poden perdre els canvis efectuats per transaccions confirmades i mai s'han de mantenir els canvis efectuats per transaccions abortades
- Situacions que posen en perill els objectius anteriors:
  - La cancel·lació voluntària d'una transacció a petició de l'aplicació
  - La cancel·lació involuntària d'una transacció
  - Una caiguda del sistema, que provocaria la cancel·lació de totes les transaccions actives
  - La destrucció total o parcial de la BD a causa de desastres o fallades de dispositius
- En funció de la situació distingim dues parts de la recuperació:
  - La **restauració**, que garanteix l'atomicitat i la definitivitat davant de cancel·lacions de les transaccions i de caigudes del sistema
  - La **reconstrucció**, que recupera l'estat de la BD davant una pèrdua total o parcial de la informació emmagatzemada a disc a causa de fallades o desastres

## Restauració

- La restauració ha de ser capaç d'efectuar dos tipus d'operacions:
  - La **restauració cap enrere**, que implica **desfer** els canvis d'una transacció abortada
  - La **restauració cap endavant**, que implica **refer** els canvis d'una transacció confirmada
- Per a desfer i refer canvis l'SGBD utilitza una estructura de dades amb informació dels canvis, el **dietari** (en anglès **log**), que guarda informació dels canvis que han efectuat les transaccions, i de les cancel·lacions i de les confirmacions d'aquestes:

T1 AA' T2 BB' T1 CC' T2 COMMIT T3 DD' T1 ABORT

## Reconstrucció

- Per a poder reconstruir l'estat d'una BD després d'una pèrdua parcial o total de les dades, cal utilitzar dues fonts d'informació:
  - Una **còpia de seguretat** (bolcat) que contingui un estat correcte de la BD
  - El contingut del dietari a partir del moment en què es va fer la còpia de seguretat
- Podem fer una classificació de les còpies de seguretat en funció de les característiques següents:
  - Les còpies de seguretat poden ser estàtiques o dinàmiques
  - Les còpies de seguretat poden ser completes o incrementals

	RU	RC	RR	S
Act Perd.	S	S	S	S
Lect No Conj.	-	S	S	S
Lect No Rep	-	-	S	S
An Inconsist.	-	S	S	
Fantasmes	-	-	X	S
Lx -- Lock de X	XX	ff	ff	ff
LS -- Lock de S	LS	No	fdr	ff
LICSX -- Lock del granul IC	LICSX	NO	NO	ff

Lx -- Lock de X

LICSX -- Lock del granul IC

fdr- fina al final read

LS -- Lock de S

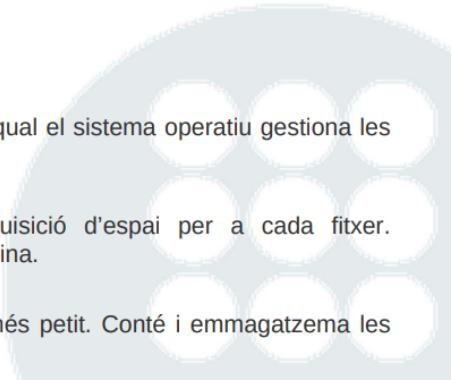
ff- fins al final

## El nivell físic

- Les dades s'emmagatzemem 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.

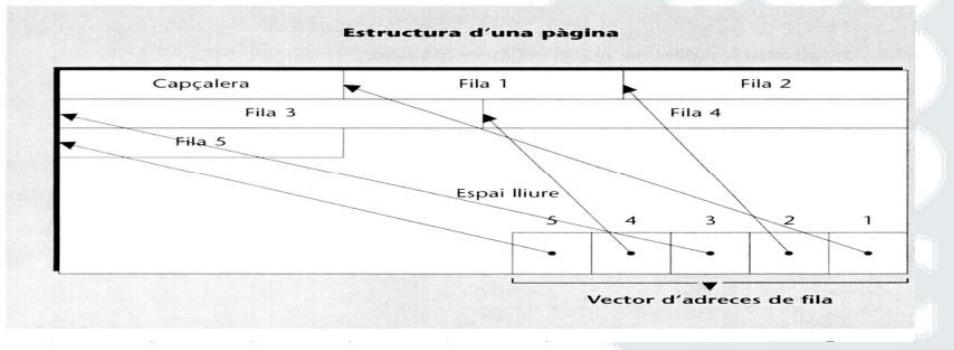
- Hi ha tres components essencials:

- 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).

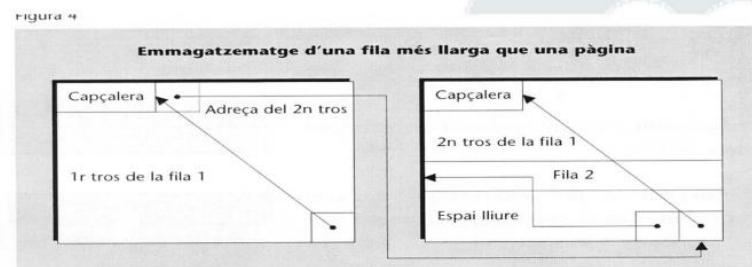


## La fila

- Estructura fila:

Capçalera	Camp1	Camp2	Camp3	...
-----------	-------	-------	-------	-----

- 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.



## El camp

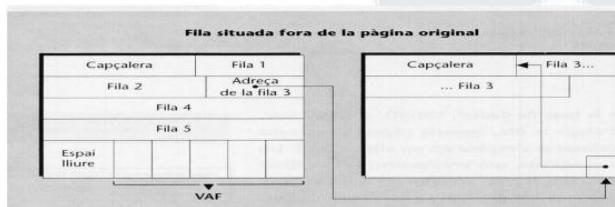
- Estructura física: 

Capçalera	Contingut
-----------	-----------
- A la capçalera del camp:
  - Ens diu si el camp és Null / not null (si el camp admets valors nuls)
  - Longitud del campSi el camp és de longitud 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 no és la longitud real del valor
  - Decimal(p,s):  $\lceil (p+1)/2 \rceil$  (és interessant comparar-lo amb float)
  - Date: aaa | 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")

Camp 1 6 bytes	Camp 2 1 bytes	Camp 3 10 bytes
1	4	Joan

## 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



## 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

## 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 voluminos → 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:

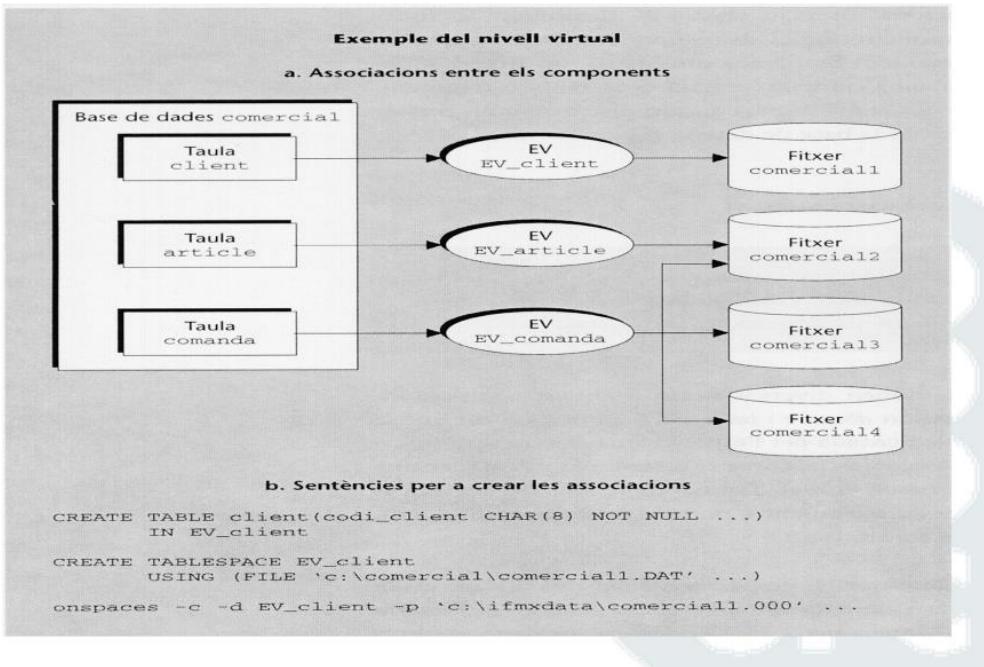
• Nom clients	varchar(60)	60 bytes màxim
• Powerpoint	transparències	6 MB
  - La representació tradicional no és adequada, així el tractament ha de ser diferent → més endavant el veurem

## EL nivell virtual: Justificació i definició

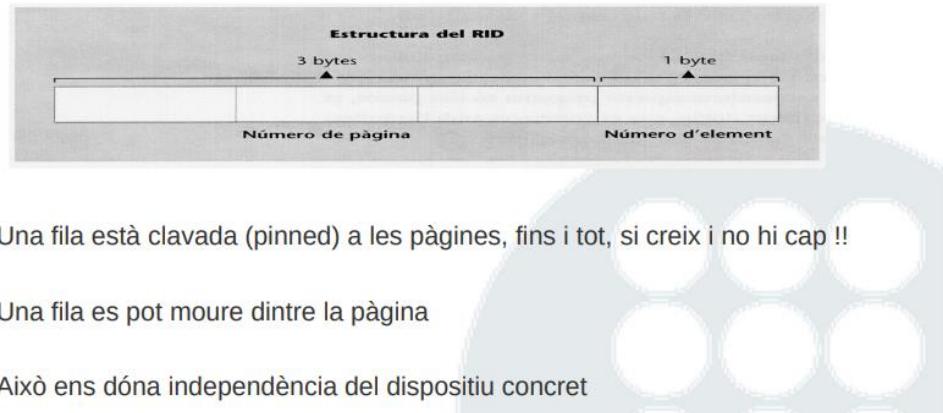
- Justificació:  
Si Taula  $\leftrightarrow$  Fitxer, no cal un nivell intermedi que faci corresponde components lògics amb físics, però
  - Si taules molt grans: hi ha fragments en dispositius diferents
  - Si taules molt petites: s'han d'agrupar en un fitxer
  - Si hi ha objectes grans
  - Si hi ha 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



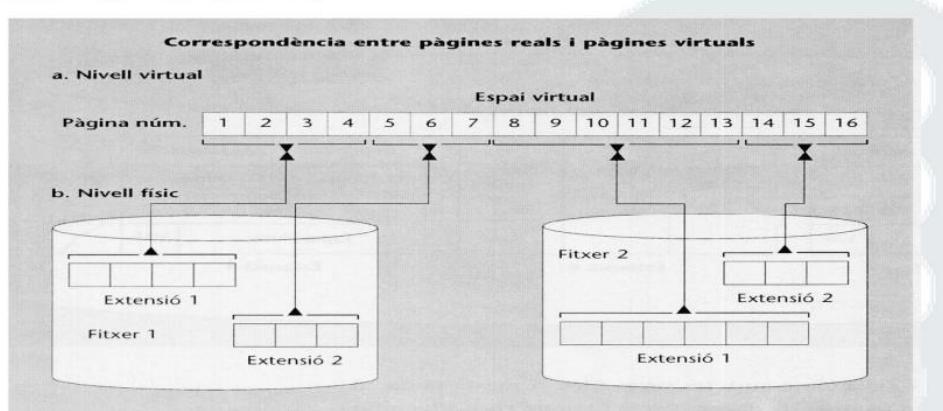
## Record IDentifier (RID)



- 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

## 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



## Tipus d'espais virtuals

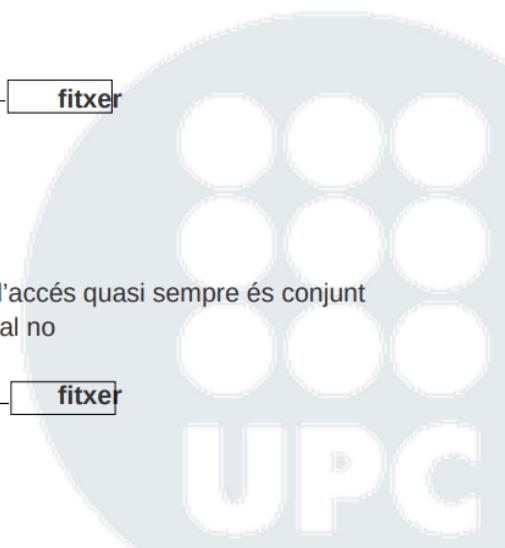
### ■ Espai de taules

- 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



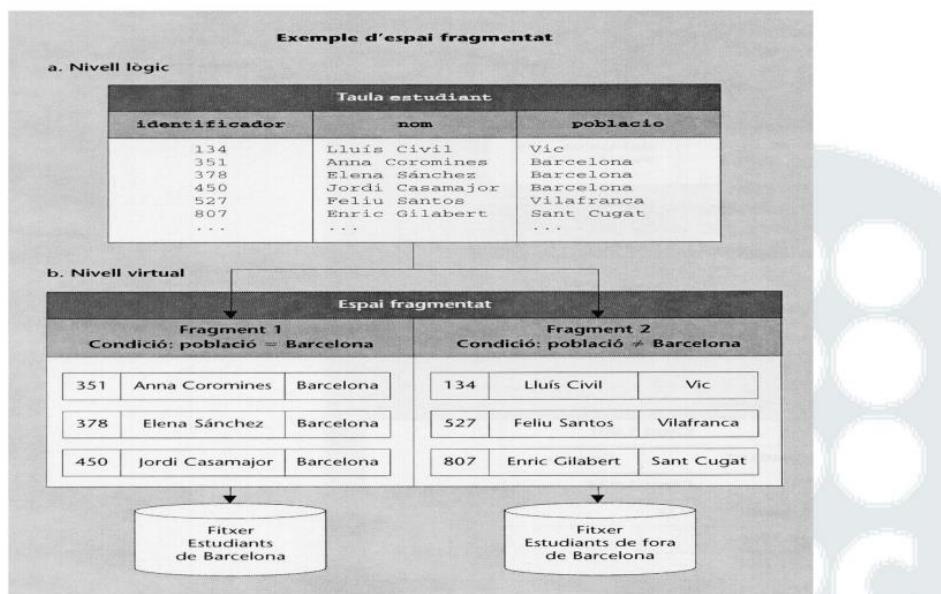
## Tipus d'espais virtuals

### Espai fragmentat

- Per a taules molt grans
  - » Ex: Info. Trucades telefòniques 2 mesos
  - » 10 milion abonats 100 trucades 2 mesos = 1000 milions files
  - » 1000 milions 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



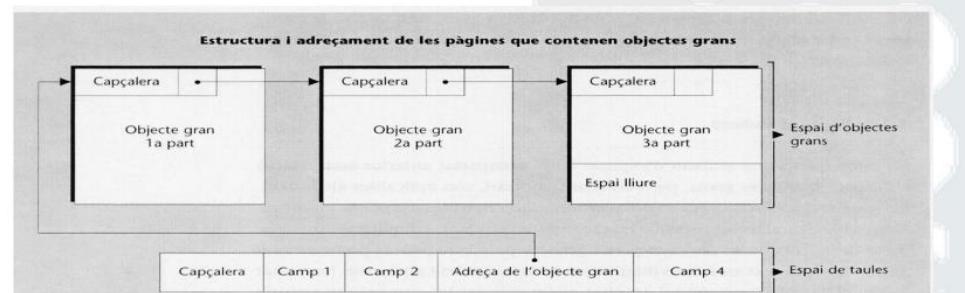
### Tipus d'espais virtuals: Exemple espai virtual fragmentat



## Tipus d'espais virtuals

### Espai d'objectes grans

- Els objectes grans s'emmagatzemem 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

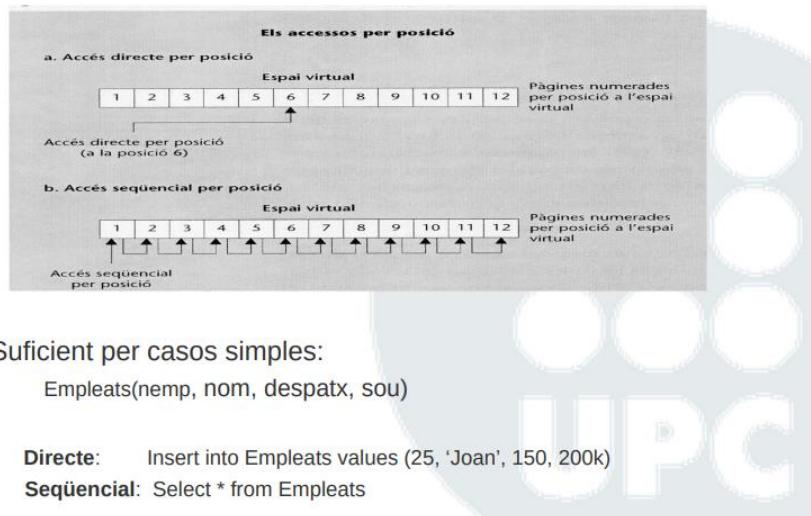


### Altres aspectes

- 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

## Mètodes d'accés: Accés per posició

- **Directa:** 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)

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

**Seqüencial:** Select \* from Empleats

## 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  
Directa | seqüencial
  - Select \* from Empleats where despatx=150 and sou=200k  
Directa | seqüencial
  - Delete from Empleats where despatx>100 and sou>180k  
Directa | seqüencial
  - Select \* from Empleats where despatx=150 and sou>200k  
Directa | seqüencial
- Exemples d'accisos 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

## Mètodes d'accés: Accés per valor

- **Directa:** 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

- **Directa:**
  - Select \* from Empleats where despatx=150
  - Update Empleats set sou=250k where despatx=200
  - Delete from Empleats where despatx=150

### Seqüencial

- **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

## 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

	Seqüència	accés seqüencial per posició	MIG	MAX
Taula	Idem		N / 2	N
Taula ord. físicament	Cerca dicotòmica Accés directe posició		$\log_2 N$	$\log_2 N + 1$
	Interpolació		$\log_2 \log_2 N$	$\rightarrow N$
Taula ord. lògicament	Cadena curta i llarga		$\sqrt{N}$	$2\sqrt{N}$

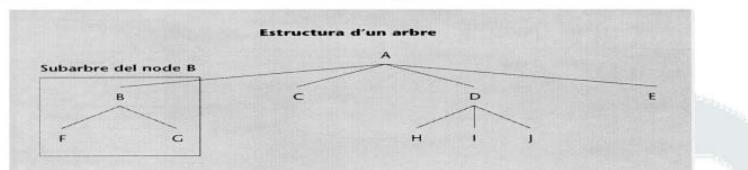
Els costos d'ambdues consultes pel cas de 300000, i 23M d'empleats seria:

- 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 !!

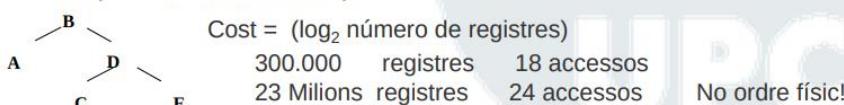
Select \* from Empleats where sou < 200000

## Arbres B+ Introducció

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

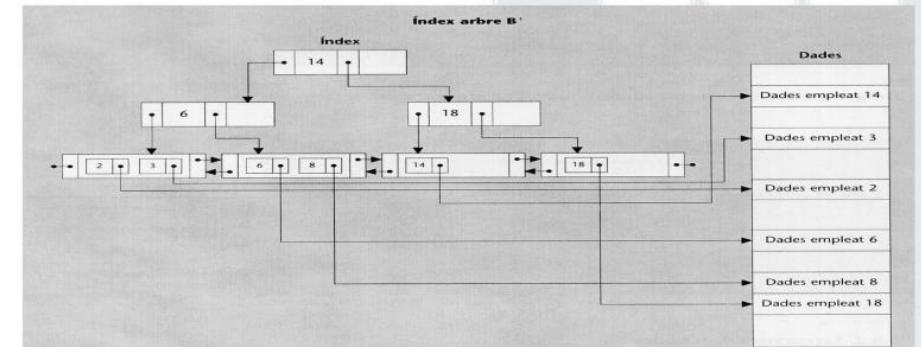


- Un arbre B+: És un tipus particular d'arbre de cerca
  - Objectiu d'aquest tipus d'arbre: Minimitzar les E/S
  - Cost arbre B+:
    - Accedir a 1 registre d'entre 300.000 amb 3 accessos a pàgines i a 1 registre d'entre 23.000.000 amb 4 accessos No ordre físic!
    - Si es compara amb el cost en el cas d'un arbre binari (arbre orientat normalment a fer cerques a memòria interna):



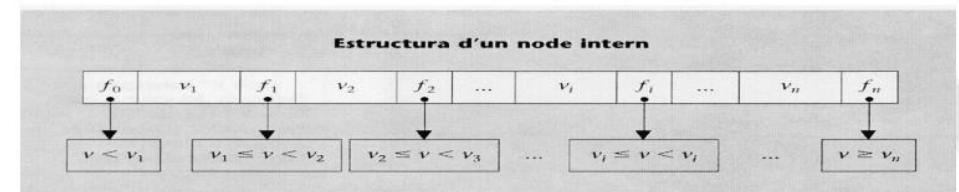
## Arbres B+ Estructura

- Tot arbre té un número d'ordre **d** que indica la capacitat dels nodes.
  - Arbre B+ 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



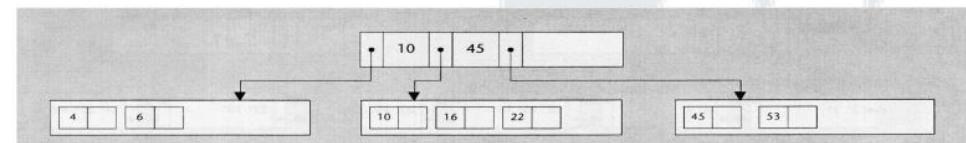
## Arbres B+ Estructura node intern

- 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

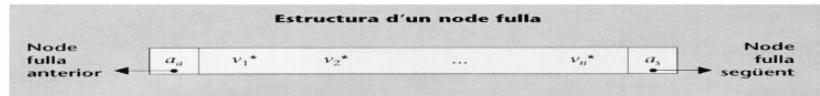
- Exemple d'ordre 2:



## Arbres B+ Estructura node fulla

- Els nodes fulla contenen:

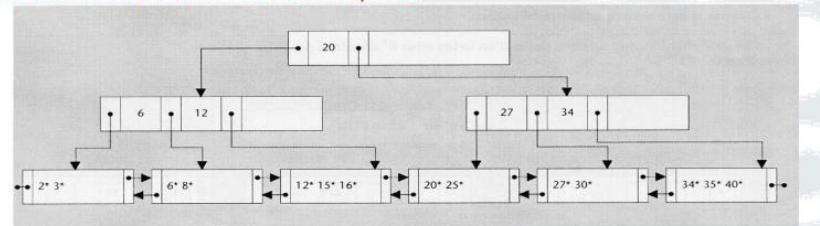
- Entrades formades per [valor, RID] que escriurem  $v^*$
- Apuntadors a la fulla anterior i següent



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.**

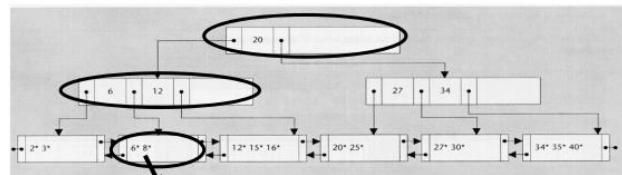


## Arbres B+ Accés directe per valor

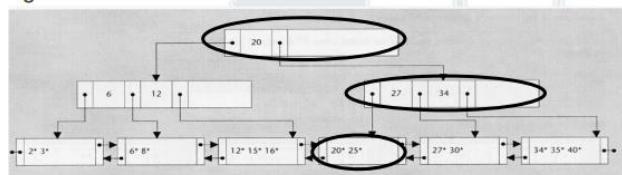
- Per accedir al registre amb valor  $v$  cal

- 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).

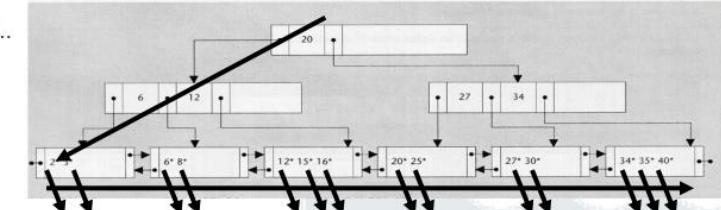
## Arbres B+ Accés seqüencial per valor

- 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:  $v_1 < \text{valor} < v_2$

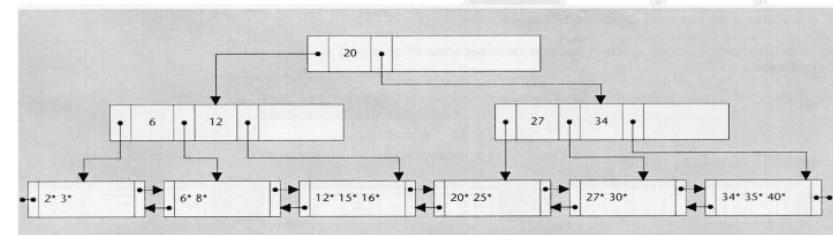
## Arbres B+ Millors de rendiment

- 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.

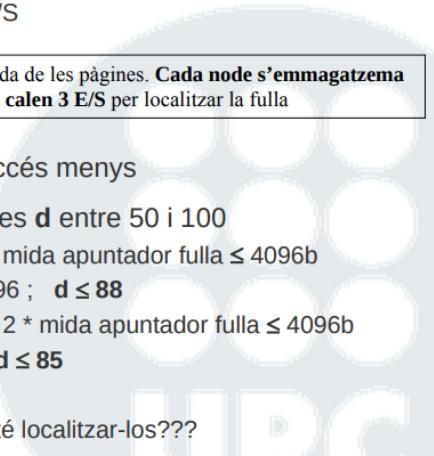
- El nombre de nodes a recórrer és sempre el mateix

## Índexs agrupats

### 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:  $2d * (\text{mida valor}) + (2d + 1) * \text{mida apuntador fulla} \leq 4096b$   
 $\Rightarrow 2d * (20) + (2d + 1) * 3 \leq 4096; d \leq 88$
  - Nodes fulla:  $2d * (\text{mida valor} + \text{mida RID}) + 2 * \text{mida apuntador fulla} \leq 4096b$   
 $\Rightarrow 2d * (20 + 4) + 6 \leq 4096; d \leq 85$
  - Ordre òptim. Simplificació: Escollim  $d = 85$
  - Quants registres podem indexar i quin cost té localitzar-los???



### Arbres B+ Exemple

- Quantes files/registres es podrien indexar amb un arbre de ordre  $d = 50$  ple al 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

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

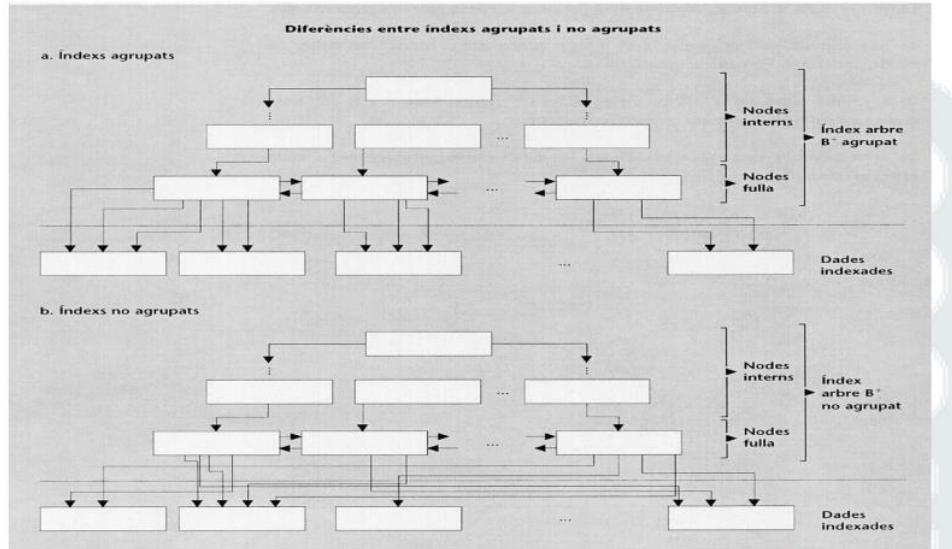
**Localitzar 1 registre entre 338100 es pot fer amb 3 accessos a pàgines de l'índex  
(+1 per accedir al registre -pàgina de dades)  
(-1 en cas que l'arrel estigui a memòria)**

- I amb 4 nivells i 69% ocupació quantes entrades?  
 $\text{Nivell 4: } 343.000 \text{ nodes amb 69 entrades} = 23667000$
- Arbre B+: càlcul de l'alçada de l'arbre

$$\approx \log_{70} 23667000 = \ln 23667000 / \ln 70 = 4;$$

$$\approx \log_{70} 338100 = 3$$

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



- 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

### Tema 9-4.1. Estructura dels Arbres B+

1. Suposa una base de dades on hi ha una taula T que té 300.000 files.

Hi ha un índex arbre B+ sobre aquesta taula definit sobre l'atribut ATR que és clau primària i és de tipus char i ocupa 6 bytes.

Els apuntadors a les pàgines de l'índex ocupen 4 bytes i els RID ocupen també 4 bytes.

Les pàgines són de 1004 bytes i el factor de bloqueig de les pàgines de dades és 25.

Determinar la D òptima d'aquest índex.

$$\begin{aligned}
 & T - 300.000 \xrightarrow{\text{D} \leq 50} \\
 & \text{B+ } (\text{ATR} / \text{PK} \rightarrow 6 \text{ bytes}) \\
 & [\text{aptr} \rightarrow 4 \text{ bytes}, \text{RID} \rightarrow 4 \text{ bytes}] \\
 & \text{pag } 1004 \text{ bytes} \\
 & n_{\text{interns}} \xrightarrow{\text{D} \leq 50} \\
 & 1004 \geq 2 * D * \text{t.valor} + 4 \\
 & (2 * D + 1) * [\text{aptr}] \\
 & n_{\text{fulla}} \xrightarrow{\text{D} \leq 49} \\
 & 1004 \geq (2 * D * \text{t.valor}) + 4 \\
 & (2 * D * \text{RID}) + 4 \\
 & 2 * [\text{aptr}] - 4
 \end{aligned}$$

### Tema 9-4.2. Estructura dels Arbres B+

2. Suposa una base de dades on hi ha una taula T que té 300.000 files.

Hi ha un índex arbre B+ sobre aquesta taula definit sobre l'atribut ATR que és clau primària.

La D de l'índex és 49, i està ple al 80%.

Quants valors hi haurà a cada node de l'índex? Quants apuntadors hi haurà a cada node de l'índex?

Quantes pàgines ocupa en total el índex?

$$\begin{aligned}
 & T - 300.000 \xrightarrow{\text{D} = 49, 80\%} \\
 & \text{ATR, PK, D} = 49, 80\% \\
 & n = 2 * D * \text{oc} = \\
 & = [2 * 49 * 0,8] = 79 \text{ valor per node} \\
 & \alpha P = n + 1 = 79 + 1 = 80 \text{ ap per node}
 \end{aligned}$$

$$\begin{aligned}
 & \text{Node arrel } 1 \leq n \leq 2d \quad \text{Quants nodes fulla?} \\
 & 37980 \text{ nodes fulla} = 3798 \text{ ap nivell ant} \quad [300000 \text{ valors} / 79 \text{ valors/node}] \\
 & 79 \text{ P ap} / 80 \text{ ap per node} = 48 \text{ nodes fulla.} \\
 & 48 \text{ ap} / 80 \text{ ap per node} = 1 \text{ node arrel}
 \end{aligned}$$

## 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<sup>+</sup>
  - 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<sub>1</sub>, v<sub>2</sub>, ...]

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
```

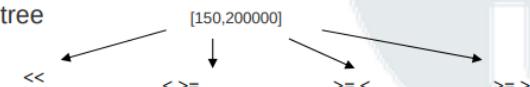
```
SELECT *
FROM Empleats
WHERE despatx = 150
AND sou > 200000
```

```
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



## 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 descendent:

```
CREATE INDEX index_despatx ON Empleats(despatx DESC)
```

### ➤ 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)
```

## Tema 9-5.1. Costos d'accés

1. Suposa una base de dades on hi ha una taula T que té 300.000 files.

Hi ha un índex agrupat arbre B+ sobre aquesta taula definit sobre l'atribut ATR que és clau primària.

La D de l'índex és 49, i està ple al 80%. El factor de bloqueig de les pàgines de dades és 20.

Calcula el cost de la consulta:  $\text{SELECT * FROM T WHERE ATR = 6789}$

Com canvia aquest cost si l'índex és no agrupat?

## Tema 9-5.2. Costos d'accés

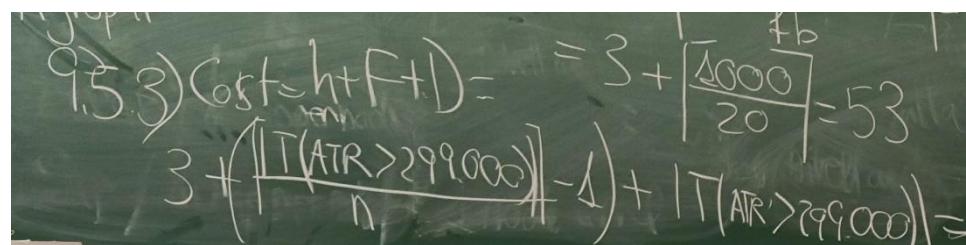
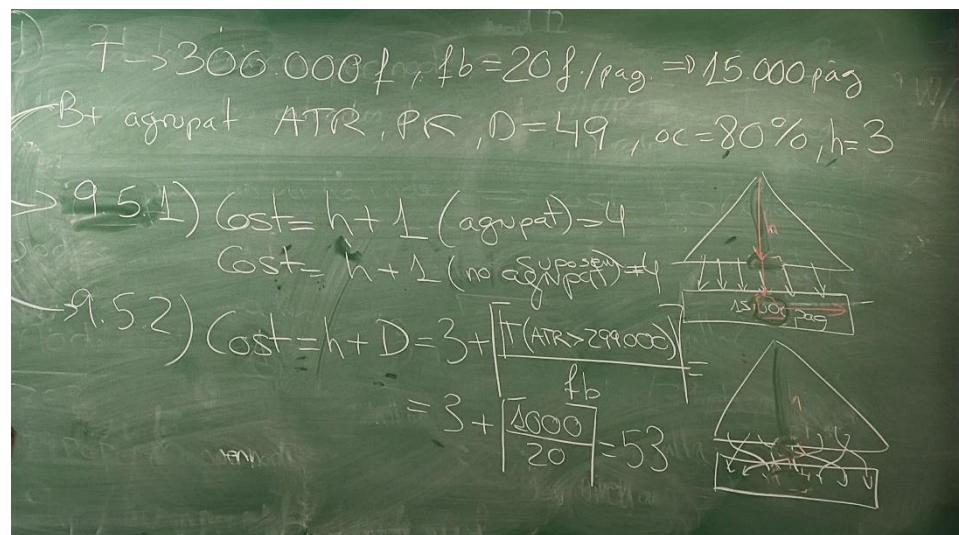
2. Suposa una base de dades on hi ha una taula T que té 300.000 files.

Hi ha un índex agrupat arbre B+ sobre aquesta taula definit sobre l'atribut ATR que és clau primària.

La D de l'índex és 49, i està ple al 80%. El factor de bloqueig de les pàgines de dades és 20.

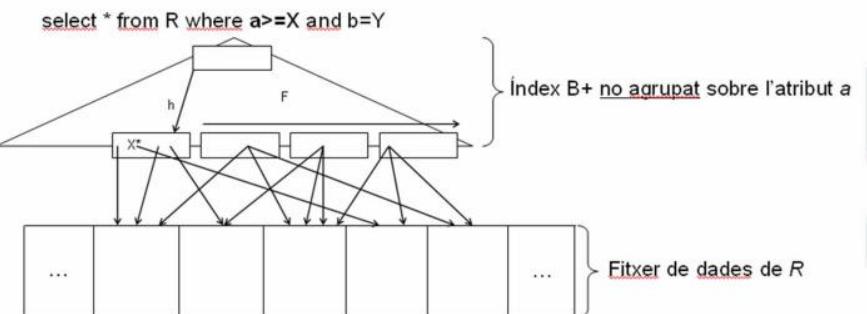
Els valors de l'atribut ATR estan distribuïts uniformement entre 1 i 300000.

Calcula el cost de la consulta:  $\text{SELECT * FROM T WHERE ATR > 299000}$



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

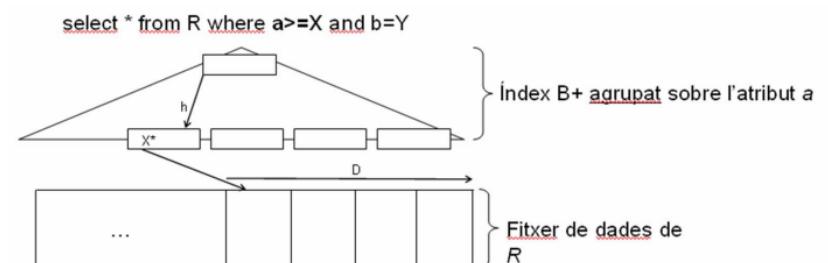
- Índex B+ no agrupat sobre un atribut a d'una taula R, l'atribut a pot prendre (o no) valors repetits



- 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
- $D = \lceil |R(a >= X)|/f \rceil$
- $|R(a >= X)|$  nombre de files de la taula R que verifiquen la condició  $a >= X$  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

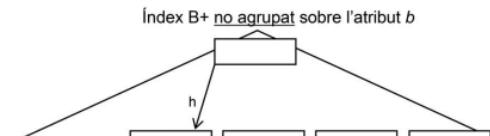
## Cost accés directe per valor: Índex arbre B+, no valors repetits

- 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  $a \geq X$  and  $b=Y$



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

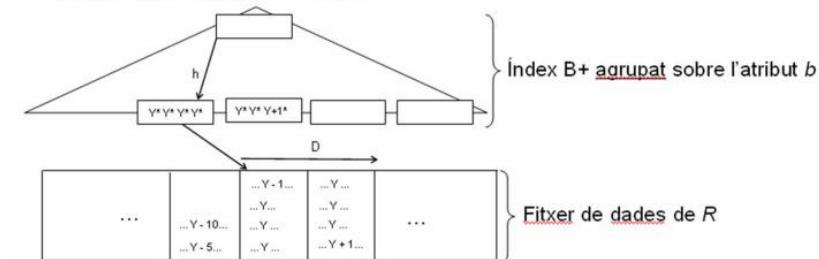


Fitxer de dades de  $R$   
(les dades no estan ordenades segons el valor de l'atribut  $b$ )

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

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

select \* from R where  $a \geq X$  and  $b=Y$

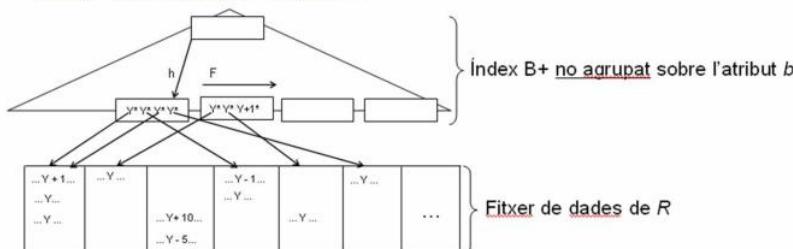


- Cost = Cost accés índex + Cost accés fitxer de dades =  $h + D$
- $h$  alçada índex B+,  $D$  nombre de pàgines del fitxer de dades que cal recórrer
- $D = \lceil |R(b=Y)|/f \rceil$
- $|R(b=Y)|$  nombre de files de la taula  $R$  que verifiquen la condició  $b=Y$  expressada a la consulta
- $f$  factor 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 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

select \* from R where  $a \geq X$  and  $b=Y$



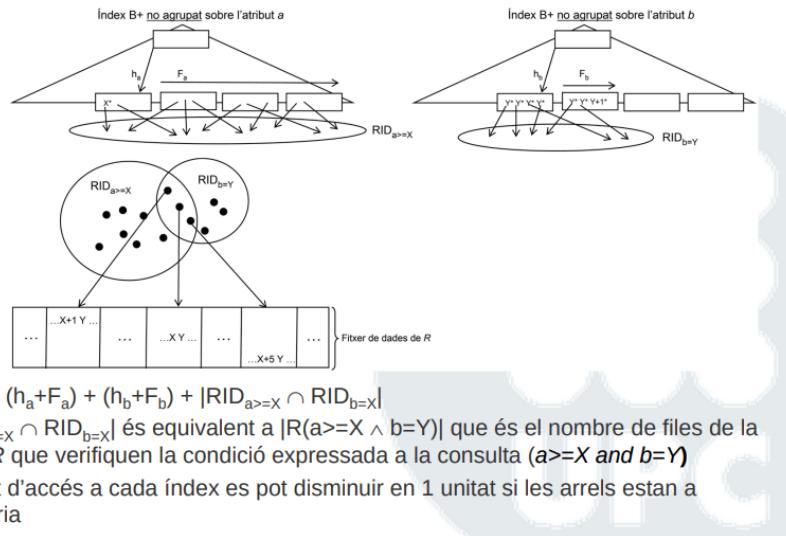
- Cost = Cost accés índex + Cost accés fitxer de dades =  $(h + F) + |R(b=Y)|$
- $h$  alçada índex B+,  $F$  nombre de nodes fulla addicionals (a més del primer node fulla) de l'índex B+ que cal recórrer
- $|R(b=Y)|$  nombre de files de la taula  $R$  que verifiquen la condició  $b=Y$  expressada a la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

## Cost accés per diversos valors: Alternatives possibles

- Imaginem que tenim la consulta següent:  
select \* from R where  $a \geq X$  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  $a \geq X$  and  $b=Y$



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

### Tema 9-6.a. Costos d'accés (ex 5.3- juny 17)

- a. Considereu la taula:

assignacions(dni, modul, numero, instantInici, instantFi)

S'emmagatzema emprant un espai virtual de taula

Té 10.000 tuples, que estan emmagatzemades en pàgines amb 10 tuples per pàgina.

Hi ha aproximadament 50 assignacions amb  $\text{dni} \geq 444$ .

Hi ha 250 assignacions al mòdul omega.

De les assignacions del mòdul 'Omega' n'hi ha 8 que són d'un  $\text{dni} \geq 444$ .

- b. Considereu la taula assignacions:

assignacions(dni, modul, numero, instantInici, instantFi)

S'emmagatzema emprant un espai virtual de taula

Té 10.000 tuples, que estan emmagatzemades en

- c. Considereu la taula assignacions:

assignacions(dni, modul, numero, instantInici, instantFi)

S'emmagatzema emprant un espai virtual de taula

Té 10.000 tuples, que estan emmagatzemades en pàgines amb 10 tuples per pàgina.

- d. Considereu la taula assignacions:

assignacions(dni, modul, numero, instantInici, instantFi)

S'emmagatzema emprant un espai virtual de taula

Té 10.000 tuples, que estan emmagatzemades en

- e. Considereu la taula assignacions:

assignacions(dni, modul, numero, instantInici, instantFi)

S'emmagatzema emprant un espai virtual de taula

Té 10.000 tuples, que estan emmagatzemades en pàgines amb 10 tuples per pàgina.

Hi ha aproximadament 50 assignacions amb  $\text{dni} \geq 444$ .

Hi ha 250 assignacions al mòdul omega.

De les assignacions del mòdul 'Omega' n'hi ha 8 que són d'un  $\text{dni} \geq 444$ .

S'han definit dos índexs B+

- L'arbre B+ per dni és d'ordre  $d=157$ , és agrupat, i té una ocupació del 70%.
- L'arbre B+ per mòdul és d'ordre  $d=227$ , és no agrupat, i té una ocupació del 60%.

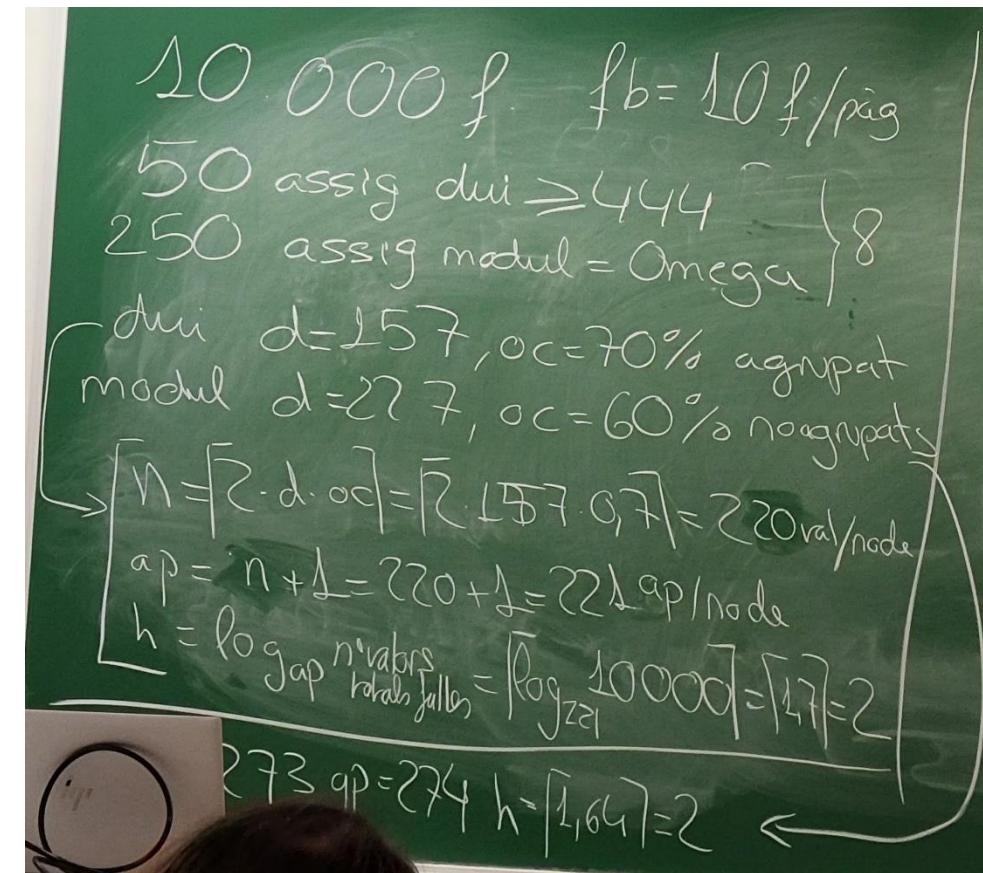
Expliqueu i mostreu com es calcula el cost de la consulta següent usant estratègia d'intersecció de RIDs:

S'han definit un índex B+

- L'arbre B+ per mòdul, dni és d'ordre  $d=100$ , és no agrupat, i té una ocupació del 80%.

Expliqueu i mostreu com es calcula el cost de la consulta següent usant l'índex:

```
SELECT count(*)
FROM assignacions a
WHERE a.dni >= '444'
      AND a.modul='Omega'
```



$$a) \text{cost} = \frac{10000}{40} = 1000 \text{ pag.}$$

b) dui, ag.

$$\begin{aligned} \text{cost} &= h + D = 2 + \left\lceil \frac{\text{assig}(\text{dui} \geq 444)}{f_b} \right\rceil \\ &= 2 + \left\lceil \frac{50}{20} \right\rceil = 7 \text{ pag} \end{aligned}$$

$$c) \text{cost} = h + F + D =$$

$$\begin{aligned} &= 2 + \left( \left\lceil \frac{\text{assig}(\text{mod} = 0m)}{n} \right\rceil - 1 \right) + \left| \text{assig}(\text{mod} = 0m) \right| \\ &= 2 + \left( \left\lceil \frac{250}{1273} \right\rceil - 1 \right) + 250 = 252 \end{aligned}$$

$$d) \text{cost} = 10000$$

$$\begin{aligned} \text{cost} &= (h_{\text{dui}} + F_{\text{dui}}) + (h_{\text{mod}} + F_{\text{mod}}) + \\ &\quad + \left| \text{assig}(\text{dui} \geq 444 \text{ and } \text{mod} = 0m) \right| < \\ &= 2 + \left( \left\lceil \frac{\text{assig}(\text{dui} \geq 444)}{n_{\text{dui}}} \right\rceil \right) + 2 + 0 + 8 = 12 \end{aligned}$$

$$n = 160$$

$$aP = 161$$

$$h = 2$$

$$\text{cost} = h + F =$$

$$= 2 + \left( \left\lceil \frac{\text{assig}(\text{dui} \geq 444 \text{ and } \text{mod} = 0m)}{160} \right\rceil - 1 \right)$$

$$= 2 + \left( \left\lceil \frac{8}{160} \right\rceil - 1 \right) = 2$$

Donat un interval de DNIs, programar un procediment emmagatzemat "llistat\_treb(dniIni,dniFi)" per obtenir la informació de **cadascun dels treballadors amb un DNI d'aquest interval**.

Per cada treballador de l'interval cal obtenir:

- Les seves dades personals: dni, nom, sou\_base i plus

- En cas que el treballador tingui 5 o més lloguers actius, al llistat hi ha de sortir una fila per cadascun dels cotxes que té llogats.

- En qualsevol altre cas, al llistat hi ha de sortir una única fila amb les dades del treballador, i nul a la matrícula.

Tingueu en compte que:

- Es vol que retorneu els treballadors ordenats per dni i matrícula de forma ascendente.

- El tipus de les dades que s'han de retornar han de ser els mateixos que hi ha a la taula on estan definits els atributs corresponents.

El procediment ha d'informar dels errors a través d'excepcions. Les situacions d'error que heu d'identificar són les tipificades a la taula missatgesExcepcions, que podeu trobar definida i amb els inserts corresponents al fitxer adjunt. En el vostre procediment heu d'incloure, on s'identifiquin aquestes situacions, les sentències:

SELECT texte INTO missatge FROM missatgesExcepcions WHERE num=\_\_\_\_; ( 1 o 2, dependent de l'error)

RAISE EXCEPTION '%',missatge;

On la variable missatge ha de ser una variable definida al vostre procediment.

Pel joc de proves que trobareu al fitxer adjunt i la crida següent,

SELECT \* FROM llistat\_treb('11111111','33333333');

el resultat ha de ser:

DNI	Nom	Sou	Plus	Matricula
22222222	Joan	1700	150	1111111111
22222222	Joan	1700	150	2222222222
22222222	Joan	1700	150	3333333333
22222222	Joan	1700	150	4444444444
22222222	Joan	1700	150	5555555555

```
exception
  when raise_exception then
    raise exception '%', sqlerrm;
  when others then
    select texte into missatge from missatgesExcepcions where num =2;
    raise exception '%', missatge;
  return;
end
$$LANGUAGE plpgsql;
```

```
create type worker as (
  dni char(8),
  nom varchar(30),
  sou_base real,
  plus real,
  matricula char(10)
);

create function llistat_treb(dni_min char(8), dni_max char(8)) returns setof worker as $$
declare
  w worker;
  missatge varchar(50);

begin
  for w in select * from TREBALLADORS where (dni >=dni_min and dni <= dni_max) order by DNI asc
  loop
    if ((select count(*) from lloguers_actius where dni = w.dni) >= 5) then
      for w.matricula in select matricula from lloguers_actius where w.dni = dni order by matricula asc
      loop
        return next w;
      end loop;
    else
      w.matricula := null;
      return next w;
    end if;
  end loop;

  if not found then
    select texte into missatge from missatgesExcepcions where num = 1;
    raise exception '%', missatge;
  end if;
end;
$$
```

Implementar mitjançant disparadors la restricció d'integritat següent:

*No es pot esborrar l'empleat 123 ni modificar el seu número d'empleat.*

Cal informar dels errors a través d'excepcions tenint en compte les situacions tipificades a la taula missatgesExcepcions, que podeu trobar definida (amb els inserts corresponents) al fitxer adjunt. Concretament en el vostre procediment heu d'incloure, quan calgui, les sentències:

SELECT texte INTO missatge FROM missatgesExcepcions WHERE num=\_\_; (el número que sigui, depenen de l'error)

RAISE EXCEPTION '%',missatge;

La variable missatge ha de ser una variable definida al vostre procediment, i del mateix tipus que l'atribut corresponent de l'esquema de la base de dades.

Pel joc de proves que trobareu al fitxer adjunt i la instrucció:

**DELETE FROM empleats WHERE nempl=123;**

La sortida ha de ser:

*No es pot esborrar l'empleat 123 ni modificar el seu número d'empleat*

```
CREATE FUNCTION e123() RETURNS trigger AS $$  
declare  
    missatge varchar(100);  
begin  
    if (old.nempl=123) then  
        SELECT texte INTO missatge FROM missatgesExcepcions WHERE num=1;  
        RAISE EXCEPTION '%',missatge;  
    end if;  
  
    if (tg_op = 'DELETE') then return old;  
    else return new;  
    end if;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER esborrat_123  
BEFORE delete or update of nempl on empleats  
FOR EACH ROW EXECUTE PROCEDURE e123();
```

Implementar mitjançant disparadors la restricció d'integritat següent:

*No es poden esborrar empleats el dijous*

Tigueu en compte que:

- Les restriccions d'integritat definides a la BD (primary key, foreign key,...) **es violen amb menys freqüència que la restricció comprovada per aquests disparadors.**

- El dia de la setmana serà el que indiqui la única fila que hi ha d'haver sempre insertada a la taula "dia". Com podeu veure en el joc de proves que trobareu al fitxer adjunt, el dia de la setmana és el 'dijous'. Per fer altres proves podeu modificar la fila de la taula amb el nom d'un altre dia de la setmana. **IMPORTANT:** Tant en el programa com en la base de dades poseu el nom del dia de la setmana en MINÚSCULES.

Cal informar dels errors a través d'excepcions tenint en compte les situacions tipificades a la taula missatgesExcepcions, que podeu trobar definida (amb els inserts corresponents) al fitxer adjunt. Concretament en el vostre procediment heu d'incloure, quan calgui, les sentències:

SELECT texte INTO missatge FROM missatgesExcepcions WHERE num=\_\_;(el número que sigui, depenen de l'error)

RAISE EXCEPTION '%',missatge;

La variable missatge ha de ser una variable definida al vostre procediment, i del mateix tipus que l'atribut corresponent de l'esquema de la base de dades.

Pel joc de proves que trobareu al fitxer adjunt i la instrucció:

**DELETE FROM empleats WHERE salari<=1000**

La sortida ha de ser:

*No es poden esborrar empleats el dijous*

```

CREATE FUNCTION borrarjueves() RETURNS trigger AS $$

declare
missatge missatgesexcepcions.texte%type;
dias dia.dia%type;
begin
select dia into dias from dia;
IF (dias = 'dijous') then SELECT texte INTO missatge FROM missatgesExcepcions WHERE num=1; RAISE EXCEPTION '%',missatge;
END IF;
return new;
END;
$$ LANGUAGE plpgsql;

```

```

CREATE TRIGGER restrict_after_empl
before delete ON empleats
FOR statement EXECUTE PROCEDURE borrarjueves();

```

En aquest exercici es tracta de mantenir de manera automàtica, mitjançant triggers, l'atribut derivat *import* de la taula *comandes*.

En concret, l'import d'una comanda és igual a la suma dels resultats de multiplicar per cada línia de comanda, la quantitat del producte de la línia pel preu del producte .

#### Només heu de considerar les operacions de tipus **INSERT** sobre la taula línies de comandes.

Pel joc de proves que trobareu al fitxer adjunt, i la sentència: **INSERT INTO líniesComandes VALUES (110, 'p111', 2);**  
La sentència s'executarà sense cap problema, i l'estat de la taula de comandes després de la seva execució ha de ser:

numcomanda	instantfeta	instantservida	numtelf	import
110	1091	1101	null	30

```

create or replace function func()
returns trigger as $$

declare PREU integer;
begin
    select p.preu INTO PREU
    from productes p
    where p.idproducte = new.idproducte;

    update comandes
    set import = import + new.quantitat * PREU
    where new.numcomanda = numcomanda;
    return new;
end;
$$LANGUAGE plpgsql;

create trigger insert_comanda
after insert on líniesComandes
for each row execute procedure func()

```

En aquest exercici es tracta definir els disparadors necessaris sobre empleats2 (veure definició de la base de dades al fitxer adjunt) per mantenir la restricció següent:

*Els valors de l'atribut ciutat1 de la taula empleats1 han d'estar inclosos en els valors de ciutat2 de la taula empleats2*

Per mantenir la restricció, la idea és que:

**En lloc de** treure un missatge d'error en cas que s'intenti executar una sentència sobre empleats2 que pugui violar la restricció,  
**cal executar** operacions compensatòries per assegurar el compliment de l'asserció. En concret aquestes operacions compensatòries ÚNICAMENT podran ser operacions DELETE.

Pel joc de proves que trobareu al fitxer adjunt, i la sentència:

**DELETE FROM empleats2 WHERE nemp2=1;**

La sentència s'executarà sense cap problema, i l'estat de la base de dades just després ha de ser:

Taula empleats1

nemp1	nom1	ciutat1
1	joan	bcn
2	maria	mad

```
CREATE FUNCTION prog() returns trigger as $$  
begin  
    if (old.ciutat1 not in  
        (select ciutat2 from empleats2)) then delete  
            from empleats1  
            where ciutat1 = old.ciutat1;  
  
    end if;  
    return new;  
END;  
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER empleats_control after update or delete ON empleats2 FOR EACH ROW  
EXECUTE PROCEDURE prog();
```

Taula empleats2

nemp2	nom2	ciutat2
2	pere	mad
3	enric	bcn

Disposem de la base de dades del fitxer adjunt que gestiona clubs esportius i socis d'aquests clubs.

Cal implementar un procediment emmagatzemat "[assignar\\_individual\(nomSoci,nomClub\)](#)".

El procediment ha de:

- Enregistrar l'assignació del soci *nomSoci* al club *nomClub*, inserint la fila corresponent a la taula *Socisclubs*.
- Si el club *nomClub* passa a tenir més de 5 socis, inserir el club a la taula *Clubs\_amb\_mes\_de\_5\_socis*.
- El procediment no retorna cap resultat.

Les situacions d'error que cal identificar són les tipificades a la taula missatgesExcepcions.

Quan s'identifiqui una d'aquestes situacions cal generar una excepció:

SELECT texte INTO missatge FROM missatgesExcepcions WHERE num=\_\_\_\_; ( 1 .. 5, depenen de l'error)

RAISE EXCEPTION '%',missatge; (missatge ha de ser una variable definida al vostre procediment)

**Suposem el joc de proves que trobareu al fitxer adjunt i la sentència**

**select \* from assignar\_individual('anna','escacs');**

La sentència s'executarà sense cap problema, i l'estat de la base de dades just després ha de ser:

Taula Socisclubs

anna	escacs
joanna	petanca
josefa	petanca
pere	petanca

Taula clubs\_amb\_mes\_de\_5\_socis

sense cap fila

```
create or replace function assignar_individual(nomSoci char(10),nomClub char(10))  
returns void as $$  
declare  
    missatge varchar(50);  
    num_homes integer;  
    num_dones integer;  
begin  
    insert into socisclubs values(nomSoci, nomClub);  
  
    num_homes = (select count(*)  
        from socis s natural inner join socisclubs sc  
        where s.sex = 'M' and sc.nclub = nomClub);  
  
    num_dones = (select count(*)  
        from socis s natural inner join socisclubs sc  
        where s.sex = 'F' and sc.nclub = nomClub);
```

```

if ((select count(*) from socisclubs s where s.nclub = nomClub) > 5 and not exists (select * from clubs_amb_mes_de_5_socis where nclub = nomClub)) then
    insert into clubs_amb_mes_de_5_socis values(nomClub);
end if;

if ((select count(*) from socisclubs s where s.nclub = nomClub) > 10) then
    SELECT texte INTO missatge FROM missatgesExcepcions WHERE num=1;
    RAISE EXCEPTION '%',missatge;
end if;

if (num_homes > num_dones) then
    SELECT texte INTO missatge FROM missatgesExcepcions WHERE num=2;
    RAISE EXCEPTION '%',missatge;
end if;
exception
when raise_exception then
    RAISE EXCEPTION '%',missatge;

when uniqueViolation THEN
    SELECT texte INTO missatge FROM missatgesExcepcions WHERE num=3;
    RAISE EXCEPTION '%',missatge;

when foreignKeyViolation THEN
    SELECT texte INTO missatge FROM missatgesExcepcions WHERE num=4;
    RAISE EXCEPTION '%',missatge;

when notNullViolation THEN
    SELECT texte INTO missatge FROM missatgesExcepcions WHERE num=4;
    RAISE EXCEPTION '%',missatge;
when others THEN
    SELECT texte INTO missatge FROM missatgesExcepcions WHERE num=5;
    RAISE EXCEPTION '%',missatge;
end;
$$ LANGUAGE plpgsql;

```

```

catch (SQLException se){
    if (se.getSQLState().equals("23505")) System.out.println("El professor ja existeix");
        else { System.out.println ("Excepcio: ");System.out.println ();
        System.out.println ("El getSQLState es: " + se.getSQLState());
        System.out.println ();
        System.out.println ("El getMessage es: " + se.getMessage());
    }
}

```

#### Execució 4

- Esborreu la fila que el programa insereix, des del DBeaver.
- Editeu el programa per tal d'implementar el bloc IMPLEMENTAR
- En aquest bloc cal implemenar en jdbc:
  - Una consulta per obtenir el dni i el nom dels professors que tenen el telèfon amb un número inferior al número de la variable buscaTelf
  - En cas que no hi hagi cap professor que tingui un telèfon amb número inferior al indicat a la variable, treure un missatge "NO TROBAT"
  - Cal mostrar amb System.out.println el resultat de la consulta.
- Executeu una altra vegada el programa
- Indiqueu quin és el resultat del select.
- Doneu el codi de la part del programa des del bloc IMPLEMENTAR fins al final.

```

String buscaTelf="3334";
ResultSet r = s.executeQuery ("select dni,nomprof " + "from professors " +
                                "where telefon < "+buscaTelf+");"

boolean found = false;
while (r.next()) {
    System.out.println(r.getString("dni") + " " + r.getString("nomprof"));
    found = true;
}
if (!found) System.out.println("NO TROBAT");

// Rollback i desconexió de la base de dades
c.commit();
c.close();
System.out.println ("Commit i desconexió realitzats correctament.");
}

catch (ClassNotFoundException ce)
{
    System.out.println ("Error al carregar el driver");
}
}}
```

- **Apartat 1**

- Editeu el programa gestioProfes per tal d'implementar el bloc IMPLEMENTAR CONSULTA
- En aquest bloc cal implemenar en jdbc:
  - Una consulta per obtenir el dni i el nom dels professors que tenen els telèfons que hi ha a l'array telfsProf.
  - En cas que hi hagi un telèfon que no sigui de cap professor caldrà que surti el número de telèfon i el text "NO TROBAT"
  - Cal mostrar amb System.out.println el resultat de la consulta.
- Executeu el programa
- Indiqueu quin és el resultat del select.

- **Apartat 2**

- Editeu el programa gestioProfes per tal d'implementar el bloc IMPLEMENTAR CANVI BD
- En aquest bloc cal implemenar en jdbc:
  - Per cada despatx del mòdul 'omega' que no té cap assignació amb instant fi null, incrementar la superfície del despatx en 3 metres quadrats.
  - Cal mostrar amb System.out.println la quantitat de files modificades.
  - En cas que la superfície d'algun dels despatxos passi a ser més gran o igual a 25, no s'ha de modificar cap despatx, i cal mostrar un missatge "Algun despatx passaria a tenir superfície superior o igual a 25".
- Indiqueu quina/es sentències SQL us ha/n fet falta per implementar el canvi, quin és el resultat de l'execució del programa, i com ho heu fet per identificar si es produeix l'excepció.

### // IMPLEMENTAR CONSULTA

```
String[] telfsProf = {"3111", "3222", "3333", "4444"};  
  
PreparedStatement ps = c.prepareStatement("select dni, nomprof " + "from professors " + "where telefon = ?");  
  
for (int i = 0; i < telfsProf.length; ++i) {  
  
    ps.setString(1, telfsProf[i]);  
  
    ResultSet rs = ps.executeQuery();  
  
    if (rs.next()) {  
  
        // retorna true --> hi ha resultat  
  
        String nom = rs.getString("nomprof");  
  
        String dni = rs.getString("dni");  
  
        System.out.println ("---TROBAT" + " ---" + dni + " ---" + nom);  
  
    } else {  
  
        // false --> no ha trobat resultat  
  
        System.out.println ("***NO TROBAT");  
  
    }  
}
```

```
// IMPLEMENTAR CANVI BD

s = c.createStatement();

int numModificades = s.executeUpdate("update despatxos d " + "set superficie = superficie + 3 " + "where modul = 'omega' and "
+ "not exists (select * from assignacions a "
+ "where instantFi is NULL and a.modul=d.modul and a.numero=d.numero)");

System.out.println ("Número de despatxos modificats" +numModificades);

System.out.println();
s.close();

// Commit i desconexió de la base de dades

c.commit();
c.close();

System.out.println ("Commit i desconexió realitzats correctament.");
}

catch (ClassNotFoundException ce) {      System.out.println ("Error al carregar el driver");  }

catch (SQLException se) {
    if (se.getSQLState().equals("23514")) System.out.println ("La superfície no pot ser més gran que 25");
    else {
        System.out.println ("Excepcio: ");System.out.println ();
        System.out.println ("El getSQLState es: " + se.getSQLState());
        System.out.println ();
        System.out.println ("El getMessage es: " + se.getMessage());
    }
}
}
```

## Comunicació usant Statement: Consultes

Suposem la taula *soci*:

```
create table socis (dni char(9),
                    nom char(30) not null,
                    telefon char(9),
                    numVISA char(12),
                    ciutatResidencia char(20) default 'Barcelona',
                    primary key (dni));
```

Obtenir tots els socis que viuen a "Barcelona":

```
Statement s = c.createStatement ();
String cr = "Barcelona";
ResultSet r = s.executeQuery ("select dni,nom "+
                             "from socis "+
                             "where ciutatResidencia = '" + cr + "'");

// Tractar el resultat
s.close();
```

## Comunicació usant PreparedStatement

Es tracta de sentències que s'executaran diverses vegades durant l'execució d'un programa. En la majoria dels casos estan parametritzades.

Els paràmetres s'especifiquen amb un signe "?" dins el *String* que forma la sentència.

Per donar valor a un paràmetre es fan servir mètodes *setXXX* que es poden invocar sobre objectes *PreparedStatement*.

```
void setXXX(int posicioParametre, XXX valor);
```

On *XXX* depèn del tipus del paràmetre, *posicioParametre* és el número d'ordre del paràmetre dins la sentència, i *valor* és el valor que hi volem assignar.

Concretament, amb els tipus que usem habitualment:

```
void setString(int posicioParametre, String valor);
void setInt(int posicioParametre, int valor);
void setNull(int posicioParametre, Types.Integer);
void setNull(int posicioParametre, Types.Char);
```

## Comunicació usant Statement: Modificacions

Suposem la taula *soci*:

```
create table socis (dni char(9),
                    nom char(30) not null,
                    telefon char(9),
                    numVISA char(12),
                    ciutatResidencia char(20) default 'Barcelona',
                    primary key (dni));
```

Modificar la ciutat de residència del soci amb dni "10":

```
Statement s = c.createStatement ();
String dni = "10";
int numTuplesModificades = s.executeUpdate ("update socis "+
                                             "set ciutatResidencia = 'Sort' "+
                                             "where dni = '" + dni + "'");

if (numTuplesModificades == 0)
    System.out.println("El soci " + dni + " no existeix");
s.close();
```

## Tractament de resultats: ResultSets

Un *ResultSet* es comporta com un cursor. Quan es crea el *ResultSet*, el cursor es troba en una fila imaginaria anterior a la primera.

- El mètode **next** - Avança el cursor a la fila següent. Retorna cert si s'ha pogut avançar, o fals si no hi ha més files.

```
boolean next()
```

- Els mètodes **getXXX** - Permeten recuperar els valors de la fila actual del cursor.

```
XXX getXXX(int numColumna)
          recupera el valor de la fila situat en la posició numColumna
XXX getXXX(String nomColumna)
          recupera el valor de la columna que té per nom nomColumna
```

	INTEGER	REAL	CHAR	...
getInt	XX	X	X	
getFloat	X	XX	X	
getString	X	X	XX	
....				

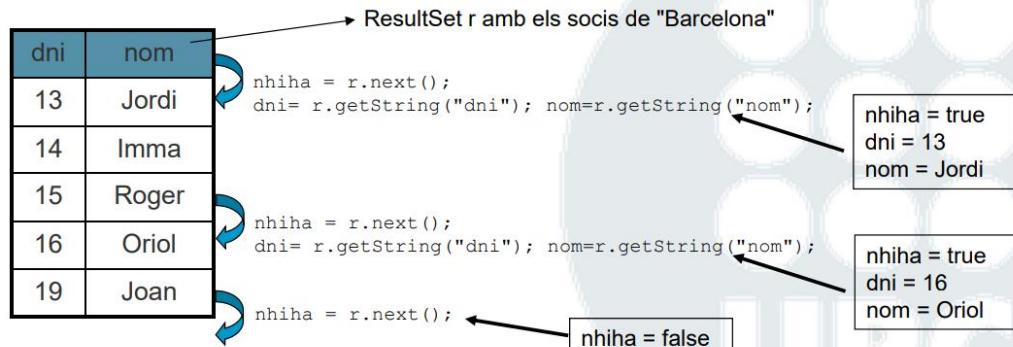
XX - el mètode és el millor per obtenir el valor  
X - el mètode permet obtenir el valor

## Exemple ResultSets

Suposem la consulta dels socis de "Barcelona":

```
ResultSet r = s.executeQuery ("select dni,nom "+  
"from socis "+  
"where ciutatResidencia = 'Barcelona';");
```

Exemple d'ús de les operacions sobre un *ResultSet*:



## Ús de Statement and PreparedStatement

- En un moment determinat només pot existir una instància de *ResultSet* per cada instància de *Statement* o *PreparedStatement*.
- És a dir, si en un cert instant necessitem accedir al resultats de dues consultes, cal que aquests resultats s'hagin obtingut en dues instàncies de *ResultSet* mitjançant dues instàncies de *Statements* i/o *PreparedStatement*.
- Quan un *Statement* o *PreparedStatement* no s'han d'usar més, es convenient de tancar-les.

```
Statement s = c.createStatement ();  
PreparedStatement ps = c.prepareStatement("select i.dni count(*) "+  
"from inscripcions i "+  
"where i.dni = ? ;");  
...  
s.close();  
ps.close();
```

## Tancament de la Connexió

- Quan sigui necessari cal fer *commit* o *rollback* del que s'ha fet fins al moment. Cal usar operacions definides sobre l'objecte de la classe *Connection* que representa la connexió amb la base de dades.

```
c.commit(); /* on c és la connexió
```

o bé

```
c.rollback(); /* on c és la connexió
```

- Finalment cal fer el tancament de la connexió. Cal usar l'operació *close* definida sobre l'objecte de la classe *Connection* que representa la connexió amb la base de dades.

```
c.close();
```

## Mètodes sobre una SQLException

Suposant que *objSE* és una instància de l'excepció *SQLException*:

Mètode	Informació que retorna
<i>objSE.getSQLState()</i>	"00000" - La sentència s'ha executat amb èxit. ">>"02XXX" - Error greu que ha impedit l'execució correcta de la sentència. "01XXX" - Situació d'avís.
<i>objSE.getMessage()</i>	Retorna un String que explica breument l'error que s'ha produït

Per exemple:

```
"23XXX" - Violació de restriccions d'integritat  
"08XXX" - Excepcions relacionades amb la connexió.  
"28000" - Autorització d'accés invalida.
```