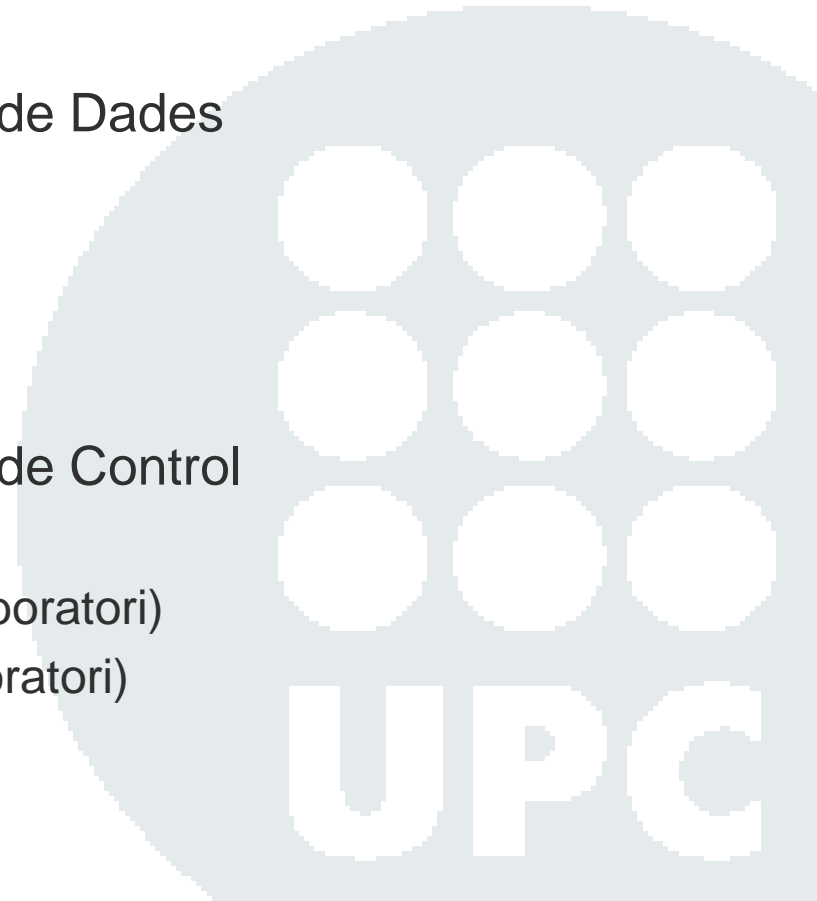


## 4. Components lògics d'una base de dades

- Objectius
- Components Lògics de Dades
  - Esquemes
  - Dominis i Taules
  - Assercions
  - Vistes
- Components Lògics de Control
  - Privilegis
  - Procediments (laboratori)
  - Disparadors (laboratori)



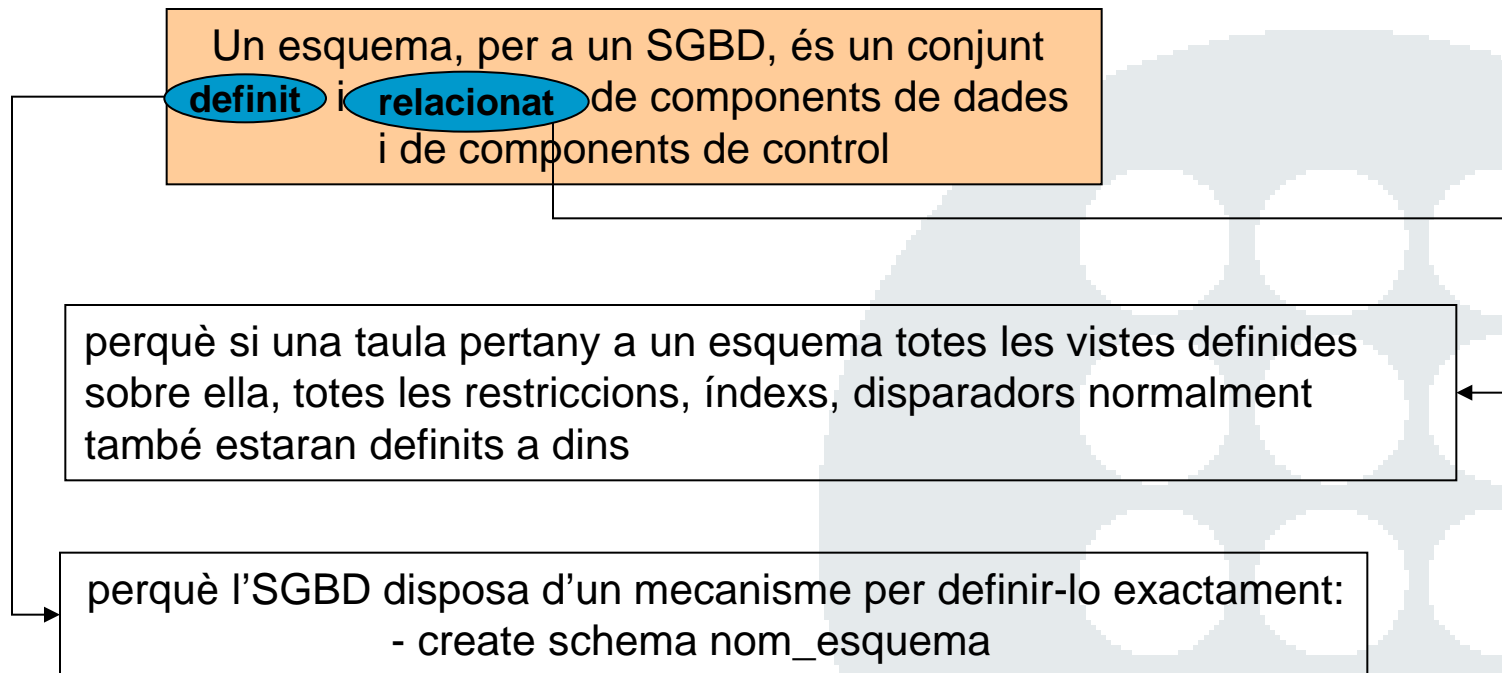
## Objectius d'aquest tema

- Completar els components lògics d'una base de dades que s'estudien a les sessions de laboratori
- Conèixer les sentències que ofereix el llenguatge estàndard SQL



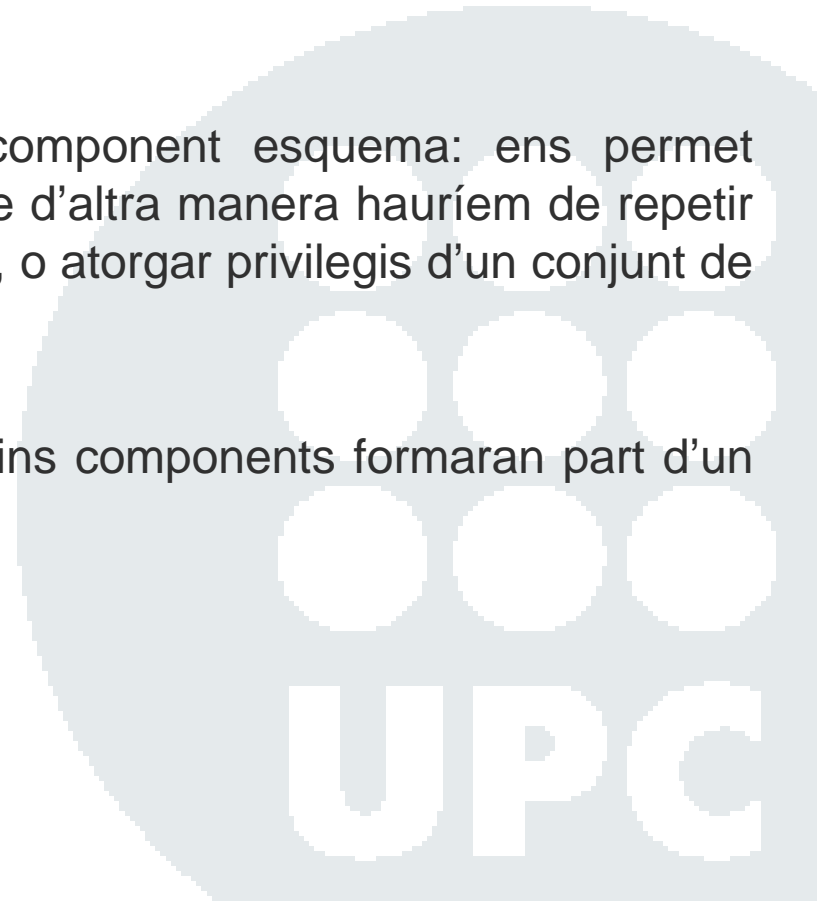
## Esquemes

Els SGBD agrupen els components de dades (taules, vistes, ...) i els components de control (procediments, disparadors, ...), que veurem més endavant, en un altre component lògic anomenat *esquema*

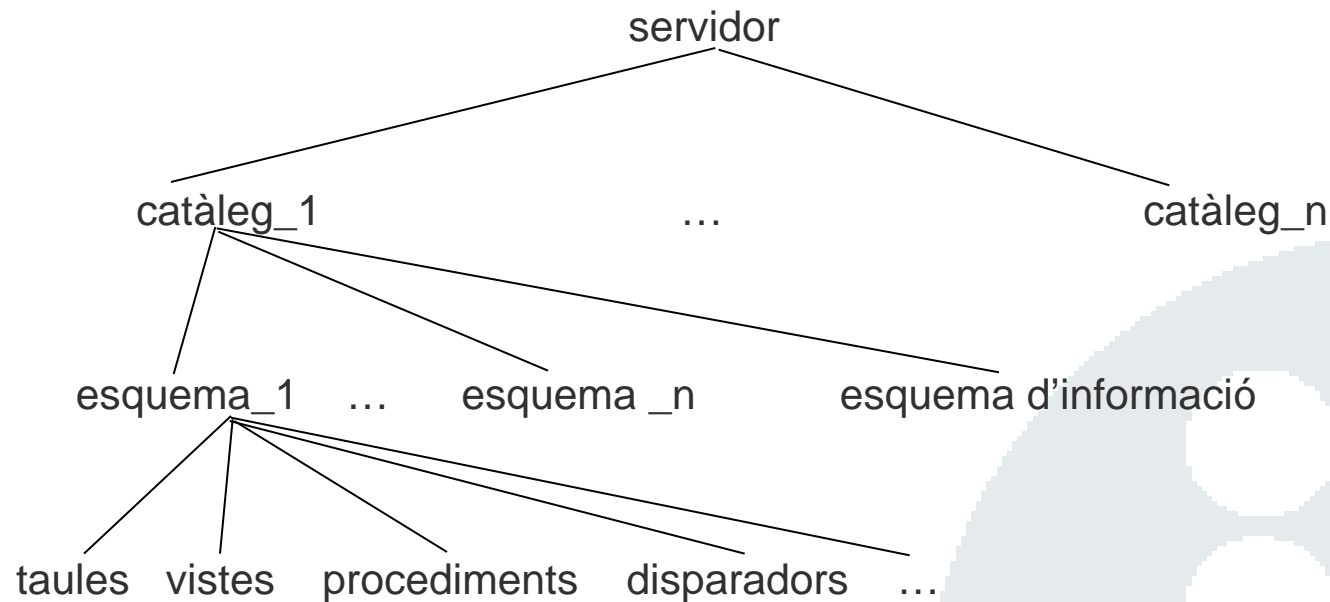


## Esquemes

- Des d'aquest punt de vista, el component esquema és una eina específica de l'SGBD que serveix com a unitat administrativa per agrupar un conjunt d'altres components.
- Aquesta és la utilitat principal del component esquema: ens permet centralitzar tasques administratives que d'altra manera hauríem de repetir individualment. Podem engegar, aturar, o atorgar privilegis d'un conjunt de components lògics a un usuari.
- Hi ha diversos criteris per a decidir quins components formaran part d'un esquema. Per exemple:
  - Per aplicacions
  - Per usuaris finals



## Servidor, Catàleg i Esquema



- Un catàleg (catalog) és un grup d'esquemes, un dels quals es anomenat esquema d'informació (information schema)
- L'esquema d'informació és un conjunt de vistes que conté la descripció de totes les dades SQL que pertanyen al catàleg corresponent
- Un servidor (cluster) pot contenir zero o més catàlegs

## Esquemes

- No hi ha una instrucció del SQL estàndar per crear, destruir o modificar Catàlegs. Dependrà de cada implementador
- La sentència **CREATE SCHEMA** dona nom a un nou esquema, identifica a l'usuari propietari de l'esquema i pot donar la llista d'elements de l'esquema

```
CREATE SCHEMA [[nom_cat.]nom_esq] [AUTHORIZATION ident_usuari]  
[llista d'elements de l'esquema];
```

- La sentència **DROP SCHEMA** amb l'opció **RESTRICT** esborra un esquema només si aquest està completament buit. En canvi amb l'opció **CASCADE** l'esborra encara que contingui elements

```
DROP SCHEMA nom_esquema [RESTRICT | CASCADE];
```

## Connexions, Sessions i Transaccions

- Una **connexió** es pot definir com l'associació que es crea entre un client SQL i un servidor SQL quan el client manifesta que té la intenció de treballar amb la BD sol·licitant una connexió.

Sentència per establir una connexió:

```
CONNECT TO nom_servidor [AS nom_connexio] [USER ident_usuari];  
SET SCHEMA nom_esq;
```

I que es destrueix quan acaba:

```
DISCONNECT nom_connexio | DEFAULT | CURRENT | ALL;
```

- Les sentències SQL que s'executen metre hi ha una connexió activa a un servidor formen una **sessió**. Per tant una sessió és el context en el que un usuari o aplicació executa una seqüència de sentències SQL mitjançant una connexió:

Sentència per establir les característiques de les transaccions que s'executen en una sessió:

```
SET SESSION CHARACTERISTICS AS mode_transac [, mode_transac ...];
```

## Connexions, Sessions i Transaccions

- Una transacció és un conjunt de sentències SQL de consulta i actualització de la BD que s'executen com una unitat. Les seves característiques es poden definir prèviament amb:

**SET TRANSACTION** mode\_transac [, mode\_transac ...];

on mode\_transac pot ser: mode\_d'accés | nivell d'aïllament

on mode\_d'accés pot ser: **READ ONLY** | **READ WRITE**

on nivell d'aïllament pot ser: **READ UNCOMMITTED** |

**READ COMMITTED** | **REPEATABLE READ** | **SERIALIZABLE** (\*)

És pot fer explícit l'inici d'una transacció amb:

**START TRANSACTION** mode\_transac [, mode\_transac ...];

- Una **transacció** finalitza confirmant o cancel·lant els canvis que s'hi han fet.

Sentència que confirma tots els canvis de la transacció:

**COMMIT [WORK] [AND [NO] CHAIN];**

Sentència que desfà tots els canvis de la transacció, deixant la BD com abans de començar l'execució de la transacció:

**ROLLBACK [WORK] [AND [NO] CHAIN];**

(\*) Els nivells d'aïllament de les transaccions s'estudiaran al tema 8 del curs



## Dominis

- Un esquema pot contenir zero o més dominis. Un domini és un conjunt de valors vàlids definits per l'usuari

```
CREATE DOMAIN nom_domini AS tipus_dades  
    [DEFAULT literal | temps | USER | NULL]  
    [llista_restriccions_domini];
```

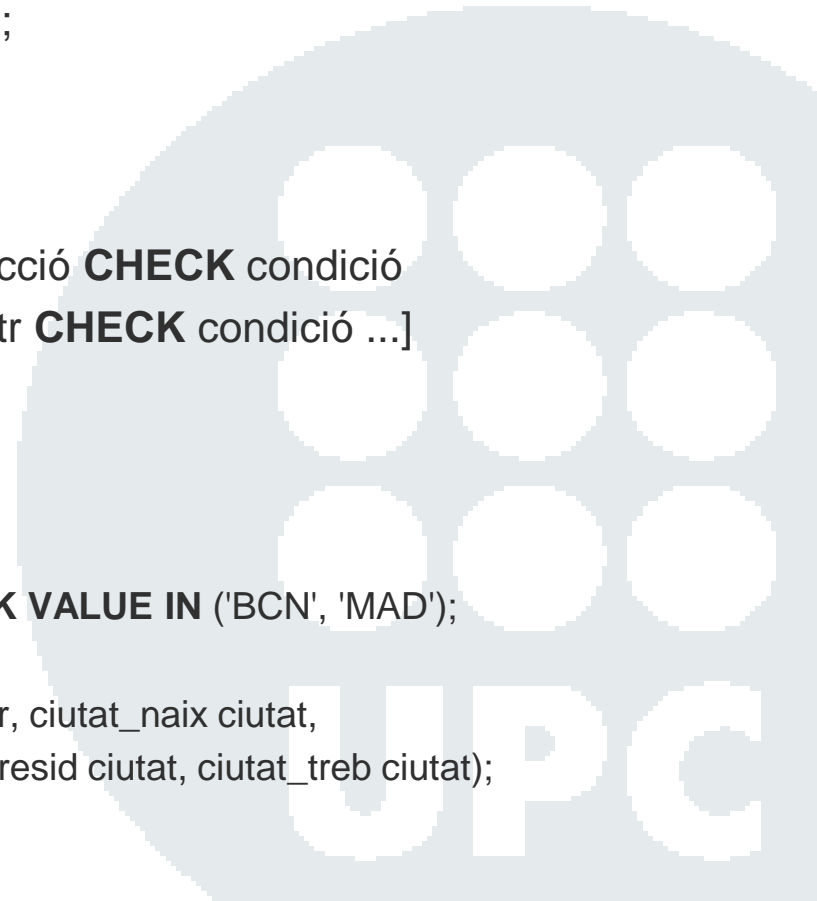
- Restriccions de domini

llista\_restriccions\_domini poden ser:

```
CONSTRAINT nom_restricció CHECK condició  
[, CONSTRAINT nom_restr CHECK condició ...]
```

```
CREATE DOMAIN ciutat AS char(15)  
    DEFAULT 'BCN'  
    CONSTRAINT vàlides CHECK VALUE IN ('BCN', 'MAD');
```

```
CREATE TABLE empleats (nempl integer, ciutat_naix ciutat,  
                        ciutat_resid ciutat, ciutat_treb ciutat);
```



# Taules

- El component lògic principal d'una base de dades és la **taula**

```
CREATE TABLE nom_taula  
  (definició_columna [, definició_columna ... ][, restriccions_taula]);
```

- Definició de columna

definició\_columna pot ser:

```
nom_columna tipus_dades [def_defecte] [restricció_columna]
```

on tipus\_dades pot ser: DATE, TIME, TIMESTAMP, INTERVAL, BINARY LARGE OBJECT(BLOB), DECIMAL (NUMERIC), FLOAT, DOUBLE PRECISION, REAL, INTEGER (INT), SMALLINT, CHARACTER (CHAR), CHARACTER LARGE OBJECT(CLOB), VARCHAR, BOOLEAN

on def\_defecte (definicions per defecte) pot ser:

```
DEFAULT literal | funció | NULL
```

on funció pot ser:

funcions de valors de temps: CURRENT\_DATE, CURRENT\_TIME,  
CURRENT\_TIMESTAMP, ...

funcions de valors d'usuaris: CURRENT\_USER, SESSION\_USER,  
SYSTEM\_USER ...

## Taules: Restriccions de columna

restricció\_columna pot ser:

- **NOT NULL**
  - La columna no pot tenir valors nuls
- **UNIQUE**
  - La columna no pot tenir valor repetits.
- **PRIMARY KEY**
  - La columna és clau primària de la taula.
- **REFERENCES** taula [(nom\_columna)]
  - [ON DELETE**  
**NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT]**  
**[ON UPDATE**  
**NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT]**
  - La columna és clau forana que referència la taula especificada (amb clau primària nom\_columna).
- **[CONSTRAINT nom\_restricció] CHECK** (condicions)
  - La columna ha de complir les condicions especificades.
  - Són típicament de rang del domini, encara que permeten posar qualsevol expressió.
  - És opcional el donar nom a aquestes restriccions.

## Taules: Restriccions de taula

restriccions\_taula poden ser:

- **UNIQUE** (columna, ....)
  - El conjunt de columnes especificades no pot tenir valors repetits.
- **PRIMARY KEY**(columna,...)
  - El conjunt de les columnes especificades és clau primària de la taula.
- **FOREIGN KEY**(columna,...) **REFERENCES** taula (nom\_columna,...)  
    **[ON DELETE**  
    **NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT]**  
    **[ON UPDATE**  
    **NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT]**
  - El conjunt de les columnes especificades és una clau forana que referencia la taula especificada (amb clau primària nom\_columna,...).
- **[CONSTRAINT nom\_restricció] CHECK** (condicions)
  - La taula ha de complir les condicions especificades.
  - La condició pot referir-se a una o més columnes de la taula.
  - És opcional el donar nom a aquestes restriccions.

Segons el estàndard SQL, en el cas que una restricció faci referència a més d'una columna, s'ha de definir com a restricció de taula.

## Taules: Restriccions

- En el cas que una restricció faci referència a un únic atribut, podem escollir si la definim com a restricció de taula o de columna.
- En el cas que una restricció faci referència a més d'una columna, s'ha de definir com a restricció de taula.
- Exemples de definició de restriccions:

```
CREATE TABLE persona (  
    Dni char(8) PRIMARY KEY,  
    Nom varchar(30) NOT NULL,  
    Data_naixement date NOT NULL  
        CONSTRAINT data_naix CHECK (data_naixement>='01/01/1900'),  
    Data_defuncio date,  
    Ciutat_naixement varchar(30) NOT NULL,  
    Ciutat_residencia varchar(30) NOT NULL,  
    Estudis char(1) DEFAULT '1'  
        CONSTRAINT estudis CHECK (estudis BETWEEN '1' and '5'),  
    Telefon decimal(9) UNIQUE,  
    CONSTRAINT dates CHECK ((data_naixement<data_defuncio) OR  
        (data_defuncio is NULL));
```

## Taules: Problemes en la definició de restriccions

- Una restricció no es viola mai si la taula està buida

- **1er Problema** (Ramakrishnan & Gehrke)

```
CREATE TABLE emp ( nemp integer,  
                    CHECK ((SELECT COUNT(*) FROM emp) > 0)
```

- Una restricció només es comprova quan s'actualitza la taula on està definida

- **2on Problema** (Garcia-Molina, Ullman & Widom)

```
CREATE TABLE emp  
  ( nemp integer,  
    dept char(10) CHECK ( dept IN SELECT dept FROM dept))
```

```
INSERT, UPDATE empleat OK  
DELETE, UPDATE dept NO
```

## Assercions

- Restriccions d'integritat que afecten a més d'una taula
- A diferència de les restriccions de columna o de taula es comproven sempre
- En la majoria dels sistemes actuals no es poden definir. Cal usar altres mecanismes: disparadors
- **CREATE ASSERTION** nom **CHECK** (condició);
- **Exemple:**

empleat( nemp, ciutat\_e, ndept) dept( ndept, ciutat\_d)

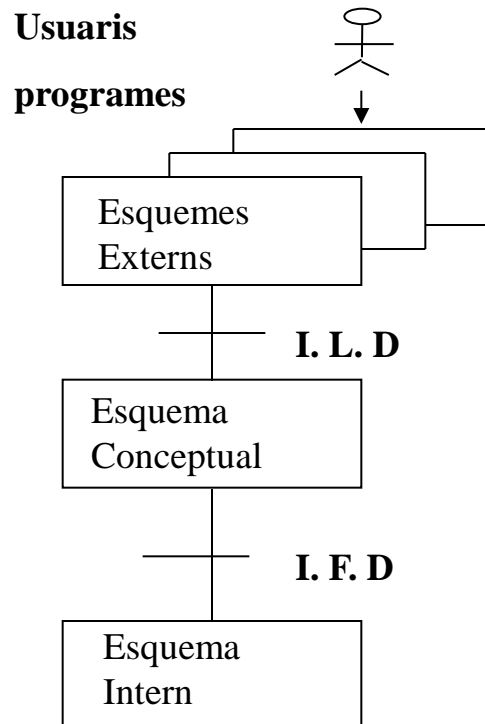
Cal assegurar que tots els empleats treballin a un departament que estigui situat a la ciutat on resideixen!

```
CREATE ASSERTION ciutat_emp_dept CHECK
  (NOT EXISTS (SELECT *
                FROM empleat e, dept d
                WHERE e.ndept = d.ndept and
                     e.ciutat_e <> d.ciutat.d));
```

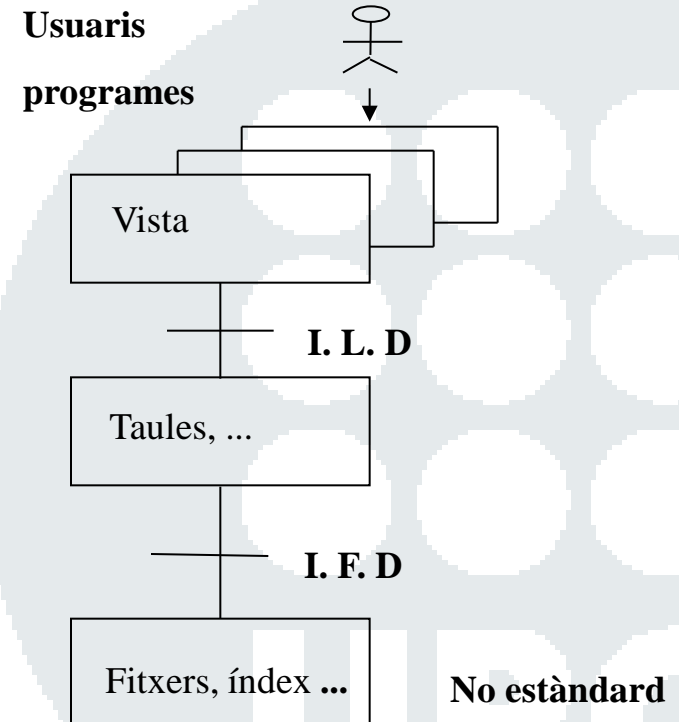
# Vistes

## Visió general

### Arquitectura ANSI/SPARC



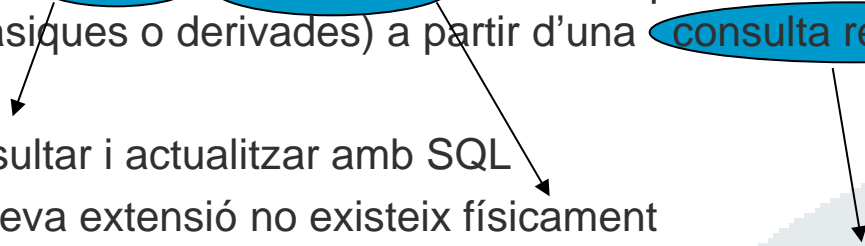
### Arquitectura Relacional





## Vistes

Una vista és una **relació derivada:** el seu esquema i contingut es deriven d'altres relacions (bàsiques o derivades) a partir d'una **consulta relacional**



Es pot consultar i actualitzar amb SQL

La seva extensió no existeix físicament

Potència SQL per definir vistes, excepte ORDER BY

**CREATE VIEW** nom\_vista [(nom\_columna, ...)] **AS** sentència\_select  
**[WITH CHECK OPTION];**

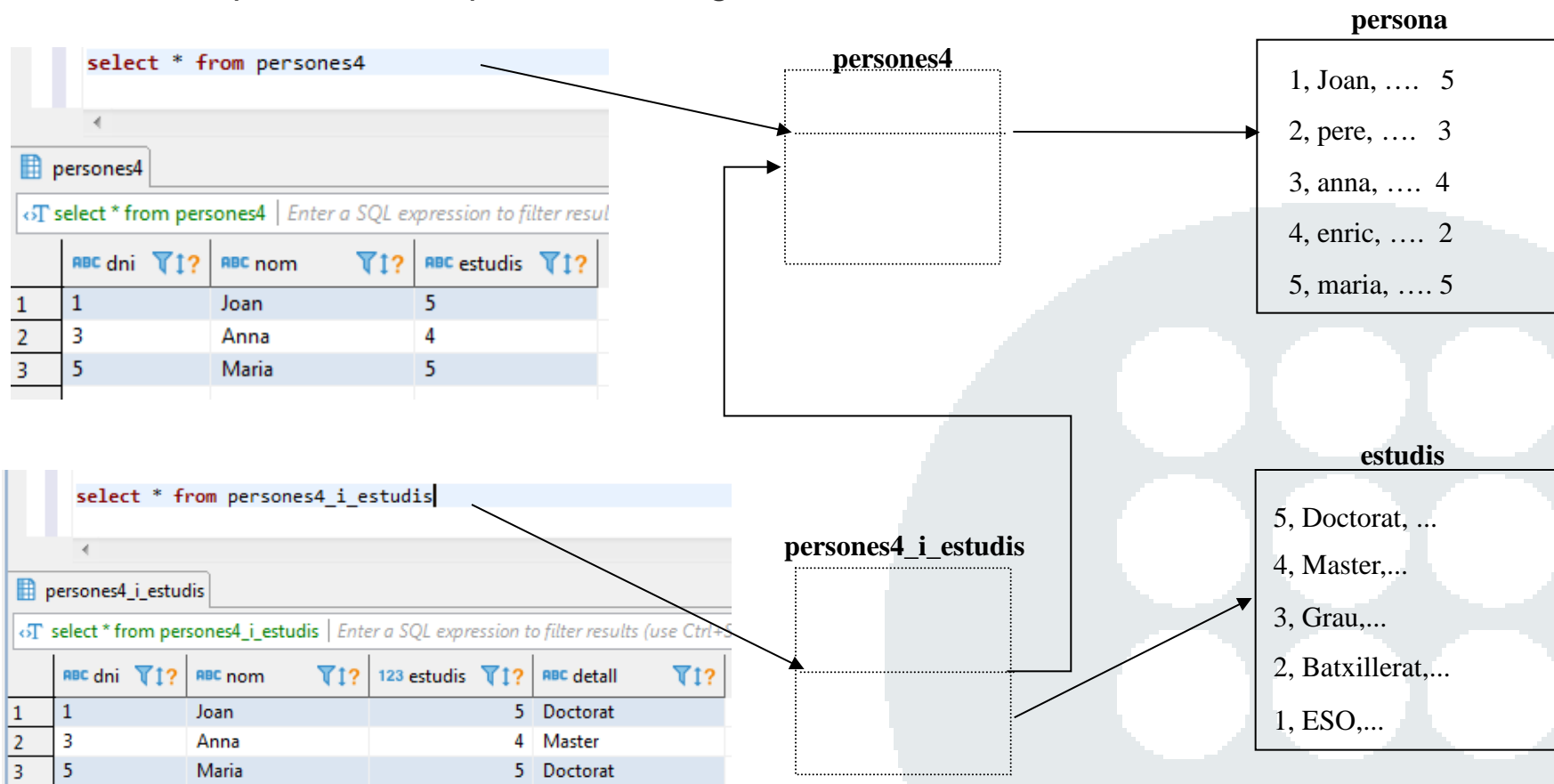
### Exemple:

persona (dni, nom, data\_naixement, data\_defuncio,  
ciutat\_naixement, ciutat\_residencia, estudis, telefon )

```
CREATE VIEW persones4 AS SELECT dni, nom, estudis
FROM persona
WHERE estudis >= 4
```

## Vistes: Consultes

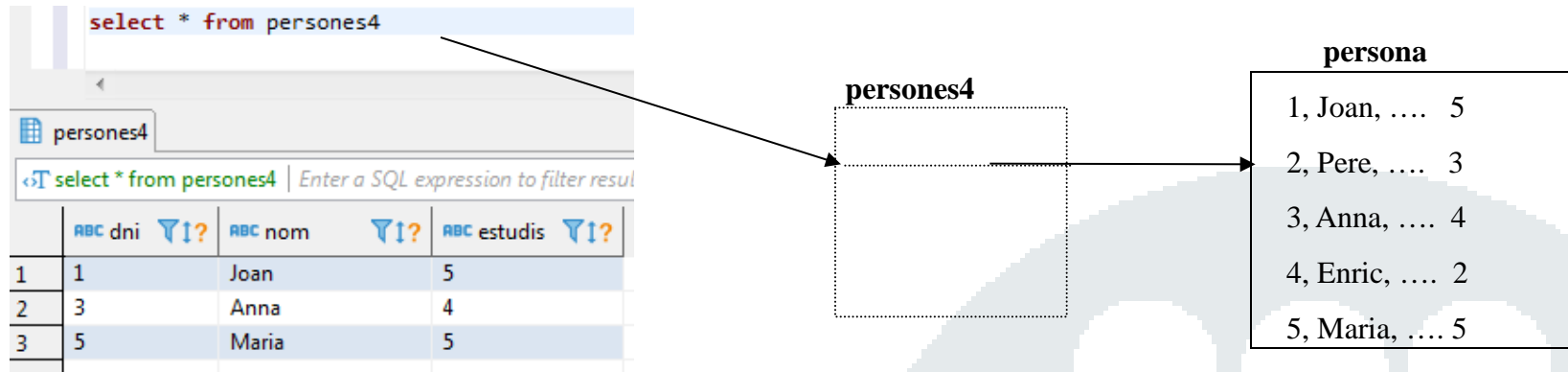
Les vistes són consultables amb SQL. De fet, de cara a l'usuari final és transparent el fet de que la relació que consulta sigui una vista



```
CREATE VIEW persones4_i_estudis AS SELECT dni, nom, estudis.estudis, estudis.detall
FROM persones4, estudis
WHERE estudis.estudis = persones4.estudis;
```

## Vistes: Actualitzacions

- Les vistes són actualitzables amb SQL. De fet, de cara al usuari final és transparent el fet que la relació que actualitzi sigui una vista



1. `UPDATE persones4 SET nom='Anna Maria' WHERE dni=5;`
2. `SELECT * FROM persones4;`

	ABC dni	ABC nom	ABC estudis
1	1	Joan	5
2	3	Anna	4
3	5	Anna Maria	5




3. `UPDATE persones4 SET estudis=2 WHERE dni = 5;`
4. `SELECT * FROM persones4;`

	ABC dni	ABC nom	ABC estudis
1	1	Joan	5
2	3	Anna	4

## Vistes: With Check Option

```
CREATE VIEW persones4 AS SELECT dni, nom, estudis
                           FROM persona
                           WHERE estudis >= 4
                           WITH CHECK OPTION;
```

1. UPDATE persones4 SET nom='Anna Maria' WHERE dni=5;
2. SELECT \* FROM persones4;

	ABC dni 	ABC nom 	ABC estudis 
1	1	Joan	5
2	3	Anna	4
3	5	Anna Maria	5

3. UPDATE persones4 SET estudis=2 WHERE dni = 5;

“ERROR: new row violates check option for view "persones4"”

## Vistes: No totes les vistes són actualitzables

- De totes maneres, no totes les vistes són actualitzables. L'estàndard defineix amb precisió quines ho són i quines no. De manera simplificada, s'admeten actualitzacions d'aquelles vistes definides com:
  - SELECT sobre una única relació R (o vista actualitzable), sense agregats ni DISTINCT
  - Els atributs del SELECT han d'incloure tots els atributs amb restriccions not null que no tinguin valor per defecte.
  - Sense GROUP BY
- El motiu de fons és que amb aquest tipus de vista qualsevol actualització de la vista sempre es pot traduir a una única actualització de la taula de base i, per tant, sense ambigüitat.
- L'estàndard expandeix les vistes actualitzables incloent certs tipus de vistes amb més d'una taula en el FROM o amb subconsultes (Ramakrishnan & Gehrke)

## Vistes: No totes les vistes són actualitzables - Exemples

subministraments( nprov, nmat, qtt)

p1    m1    100

p2    m1    200

p2    m2    200

```
CREATE VIEW sumaqtt(material,suma) AS SELECT nmat, sum(qtt)
FROM subministraments
GROUP BY nmat
```

1. DELETE de “m1,300” de la vista => DELETE “p1, m1, 100”  
DELETE “p2, m1, 200”

**Una única solució! Hauria de ser possible traduir-la.**

Però no ho és ni a l'estàndard, ni als SGBD concrets

2. UPDATE de “m1, 300” per “m1, 301” =>

**Com es tradueix? Hi ha múltiples solucions !!!**

## Vistes: No totes les vistes són actualitzables - Exemples

proveidors(nprov, nom, ...)  
100 joan

subministraments(nprov, nmat, qtt,...)  
100 m1 5230

```
CREATE VIEW prov_subprov AS SELECT *  
FROM proveidors, subministraments  
WHERE proveidors.nprov = subministraments.nprov
```

1. DELETE “100, joan,...m1,...” de la vista =>
  - 1.1 DELETE “100, m1, 5230, ...” de subministraments
  - 1.2 DELETE de proveidors, DELETE de subministraments
  - 1.3 DELETE “100, joan, ....” de proveidors
  - ....

**Múltiples solucions!!! No és possible**

2. INSERT “200, pere, ..., 200, m2, ...” a la vista =>

INSERT “200, pere, ...” a proveidors, INSERT “200, m2, ...” a subministraments

**Una solució! Hauria de ser possible**

## Esquema d'Informació

- Cada catàleg conté un esquema d'informació (*Information Schema*), a més dels esquemes definits pel propi usuari
- Tota la informació dels esquemes definits pels usuaris: noms i atributs de les taules, índexs, restriccions de columna, taula,... s'emmagatzema a la seva vegada a l'esquema d'informació. Així l'esquema d'informació és un esquema sobre esquemes: meta-dades !!
- L'esquema d'informació consisteix en un conjunt de vistes accessibles pels usuaris:
  - SCHEMATA: Informació de cada esquema del catàleg
  - DOMAINS: Informació sobre els dominis
  - TABLES: Informació sobre les taules
  - VIEWS: Informació sobre vistes
  - ASSERTIONS: Informació sobre restriccions
  - TRIGGERS: Informació sobre disparadors
  - ...
- Les vistes de l'esquema d'informació estan definides sobre un conjunt de *taules de sistema (definition schema)* accessibles només per l'administrador
- Altes, baixes i modificacions en aquestes taules de sistema són indirectes!!!



## Privilegis

- Una Base de Dades té molts objectes, molts usuaris i molts grups d'usuari. No tots els usuaris han d'accedir a tots els objectes. L'SGBD ha d'establir un mecanisme de control d'accés dels usuaris sobre aquests objectes.
- Aquest mecanisme es basa en el concepte de PRIVILEGI:

L'autorització que es dona a un  
**usuari / grup d'usuaris**  
per realitzar una  
**operació**  
sobre un  
**objecte** d'un esquema

- Els privilegis s'assignen i es revoquen amb les sentències GRANT i REVOKE.



## Privilegis

- SQL defineix 9 tipus de PRIVILEGIS:

- SELECT
  - INSERT
  - UPDATE
  - DELETE
  - REFERENCES
  - USAGE
  - TRIGGER
  - EXECUTE
  - UNDER
  - ALL
- Poden tenir associada una llista d'atributs
- Aplicables a una taula o vista
- És el dret a fer referència a una taula en una restricció d'integritat
- És el dret a utilitzar altres objectes en les pròpies declaracions
- És el dret a definir disparadors sobre una taula
- És el dret a executar una peça de codi, per exemple procediments
- És el dret a crear subtipus d'un tipus donat

## Privilegis

- Quan un usuari crea un esquema s'identifica amb la clàusula **AUTHORIZATION** i té tots els privilegis sobre ell. L'esquema serà inaccessible a altres usuaris fins que el propietari d'aquest esquema autoritzi privilegis a altres usuaris amb la sentència **GRANT**.
- Quan una sessió es comença amb una connexió tenim l'oportunitat d'indicar l'usuari amb la clàusula **USER**

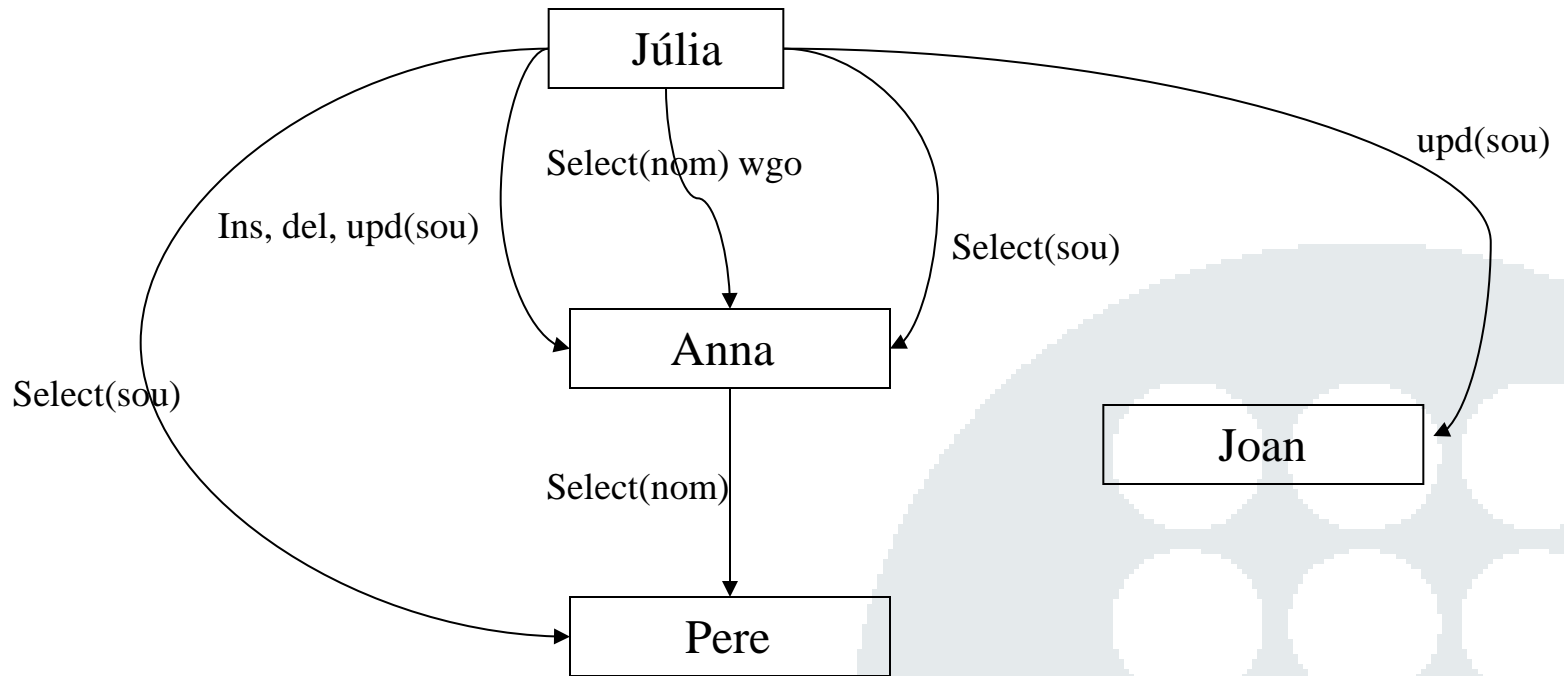
**CONNECT TO** nom\_servidor **AS** nom\_connexio **USER** ident\_usuari

- Per tant, podem executar una operació SQL només si l'usuari identificat té tots els privilegis necessaris per fer l'operació sobre els objectes involucrats

## Privilegis: Sentències GRANT i REVOKE

- Autoritzant privilegis:  
**GRANT** privilegis **ON** objectes **TO** usuaris [**WITH GRANT OPTION**];
- Revocant privilegis:  
**REVOKE** [**GRANT OPTION FOR**] privilegis **ON** objectes **FROM** usuaris {**CASCADE** | **RESTRICT**};
- **CASCADE**: Es revoquen també tots els privilegis concedits a partir dels privilegis revocats, excepte que estiguin autoritzats per una altra via
- **RESTRICT**: No es permet revocar el privilegi si amb **CASCADE** es revoqués algun altre privilegi
- Exemple:  
**Júlia**:  
CREATE TABLE empleats (n integer, nom char(10), sou integer);  
GRANT insert, delete, update(sou) ON empleats TO Anna;  
GRANT select(nom) ON empleats TO Anna WITH GRANT OPTION;  
GRANT select(sou) ON empleats TO Anna, Pere;  
GRANT update(sou) ON empleats TO Joan;  
**Anna**:  
GRANT select(nom) ON empleats TO Pere;  
**Joan**:  
UPDATE empleats SET sou=4;

## Privilegis: Diagrama d'autoritzacions



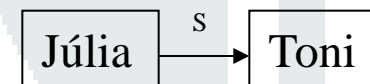
**Júlia:** GRANT insert, delete, update(sou) ON empleats TO anna;  
GRANT select(nom) ON empleats TO anna WITH GRANT OPTION;  
GRANT select(sou) ON empleats TO anna, pere;  
GRANT update(sou) ON empleats TO joan;

**Anna:** GRANT select(nom) ON empleats TO pere;

**Júlia:** REVOKE select(nom) ON empleats FROM anna CASCADE;

## Privilegis: Moltes Subtiletes

- Si la Júlia en lloc de GRANT select(sou) ON empleats TO Pere hagués fet  
GRANT select(nom) ON empleats TO Pere  
després del revoke el pere conservaria el privilegi select(nom)
- Si la Júlia en lloc de GRANT select(sou) ON empleats TO Pere hagués fet  
GRANT select ON empleats TO pere  
després del revoke el pere conservaria el privilegi select
- El Joan necessita privilegis diferents per:  
UPDATE empleats SET sou=4;  
UPDATE empleats SET sou=sou-1;
- Júlia: GRANT select ON empleats TO Toni WITH GRANT OPTION;  
Toni: GRANT select ON empleats TO Enric;  
Júlia: REVOKE GRANT OPTION FOR select FROM empleats TO Toni CASCADE;



## Privilegis i Vistes

- Exemple:

Donada una taula empleats (nemp, adreça, sou) es vol que l'empleat Joan només pugui veure el seu sou.

GRANT + VISTES = precisió en les autoritzacions

```
CREATE VIEW sou_joan AS  
  SELECT sou FROM empleats WHERE nemp="joan";  
GRANT SELECT ON Sou_joan TO joan;
```

- Els privilegis que es poden donar sobre una vista són els mateixos que es poden donar a una taula: SELECT, INSERT, UPDATE, DELETE

## Privilegis: ROLS

- Nombre elevat de GRANT per implantar la BD !!
- **ROL:** és una agrupació de privilegis definida per a un grup d'usuaris específics.
  - permet al DBA estandarditzar i canviar els privilegis de molts usuaris tractant-los com a membres d'una classe
  - Similar al concepte de grup dels SO

- **DBA :**

CREATE ROLE lector

GRANT select ON T1 TO lector

GRANT select ON T2 TO lector

CREATE ROLE escriptor;

GRANT update ON T1 TO escriptor

GRANT update ON T2 TO escriptor

GRANT lector TO escriptor

- **DBA o un usuari amb grant option**

GRANT lector TO Joan, Anna WITH GRANT OPTION

GRANT escriptor to Toni

- **DBA:** Manipulació dinàmica de privilegis

GRANT select on T3 to lector

GRANT update on T3 to escriptor

REVOKE update on T2 from escriptor CASCADE

GRANT update(a) on T2 to escriptor

- **Usuari:** Han d'activar els rols

Anna: SET ROLE lector

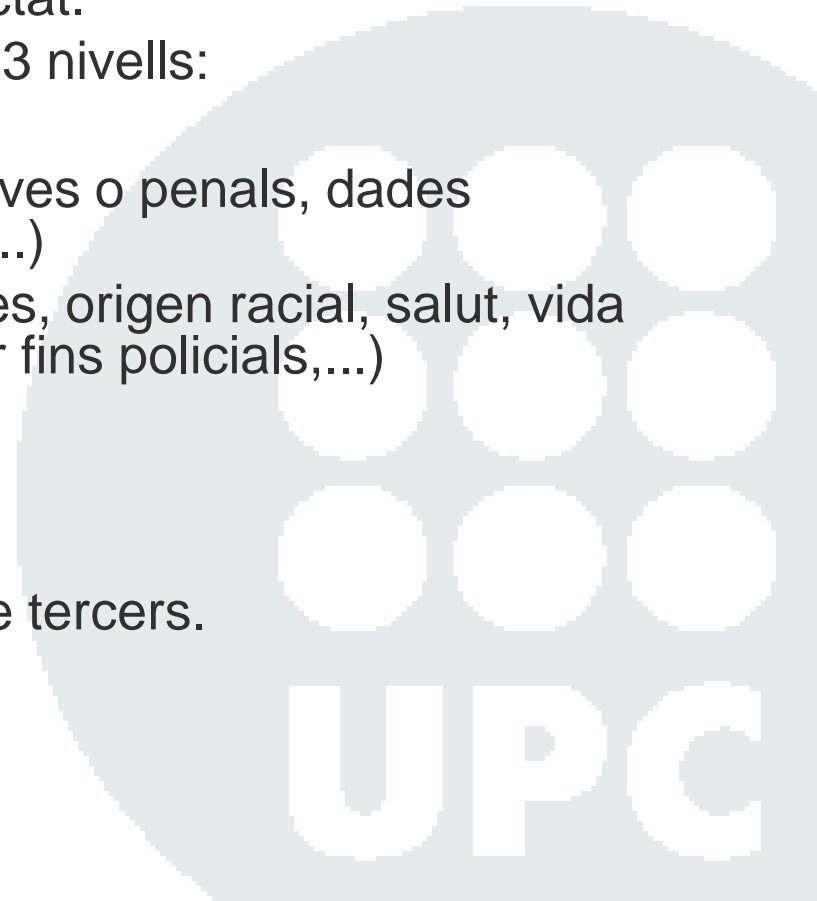


## Llei orgànica de protecció de dades personals (LOPD)

- Llei orgànica 15/1999, 13 de desembre, de protecció de dades de caràcter personal (LODP)
- Real decret 994/1999, 11 de juny.
  - *Defineix principalment els nivells de seguretat segons els tipus de dades personals*
- *Modificació en el BOE núm. 55, de 5 de març de 2011, dins la Llei d'economia sostenible. Els canvis entre altres són:*
  - *Redefinició nivells de gravetat de les infraccions.*
  - *Canvis de les sancions.*
- Llei catalana 32/2010, 1 d'octubre.
  - Defineix les competències de l'Autoritat Catalana de Protecció de Dades (APDCAT). Protecció de dades en l'àmbit de les administracions públiques de Catalunya

## LOPD: Principis de la protecció de dades

- Utilització de les dades
- Dret d'informació en la recollida de dades.
- Dret a l'accés, actualització i esborrat de dades.
- Consentiment inequívoc de l'afectat.
- Dades especialment protegides. 3 nivells:
  - Bàsic (dades personals)
  - Mitjà (infraccions administratives o penals, dades d'hisenda, serveis financers,...)
  - Alt (ideologia, religió, creences, origen racial, salut, vida sexual, dades obtingudes per fins policials,...)
- Seguretat de les dades
- Deure de secret
- Comunicació de les dades
- Accés a les dades per compte de tercers.

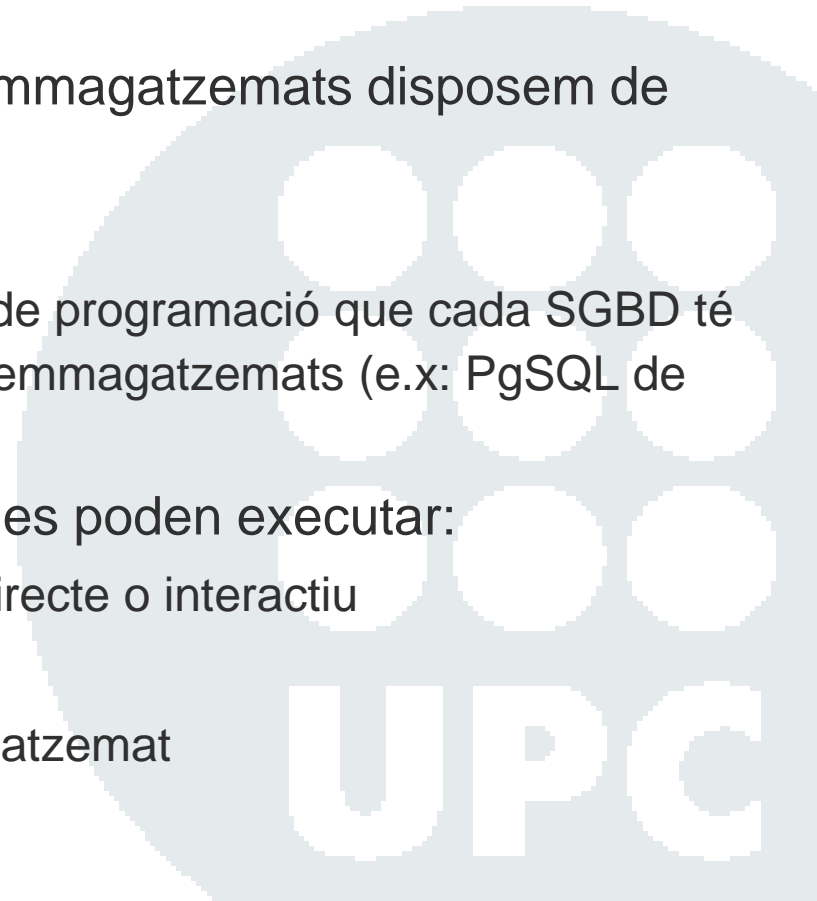


## General Data Protection Regulation (GDPR)

- Nou reglament (UE) del 27 d'abril de 2016 relatiu a la protecció de les persones físiques en el que respecta al tractament de dades personals i a la lliure circulació d'aquestes dades. *Diari Oficial de la Unió Europea 2016/679*
- Entrada en vigor: 25 de maig de 2018
- Principals canvis respecte a la LOPD:
  - *Sancions més dures: 20 milions d'euros o el 4% de la facturació anual.*
  - *Dades de caràcter personal, i dades especialment protegides (salud, ideologia, religió, origen racial, vida sexual, infraccions, genètiques, biomètriques).*
  - *Introducció de la figura del Delegat de protecció de dades.*
  - *Apareixen altres drets: dret a l'oblit, dret de supressió, etc.*
  - *Se suprimeix l'acceptació tàcita (només hi haurà consentiment inequívoc o consentiment explícit).*
  - *Apareix el concepte d'anàlisi del risc en la protecció de les dades.*

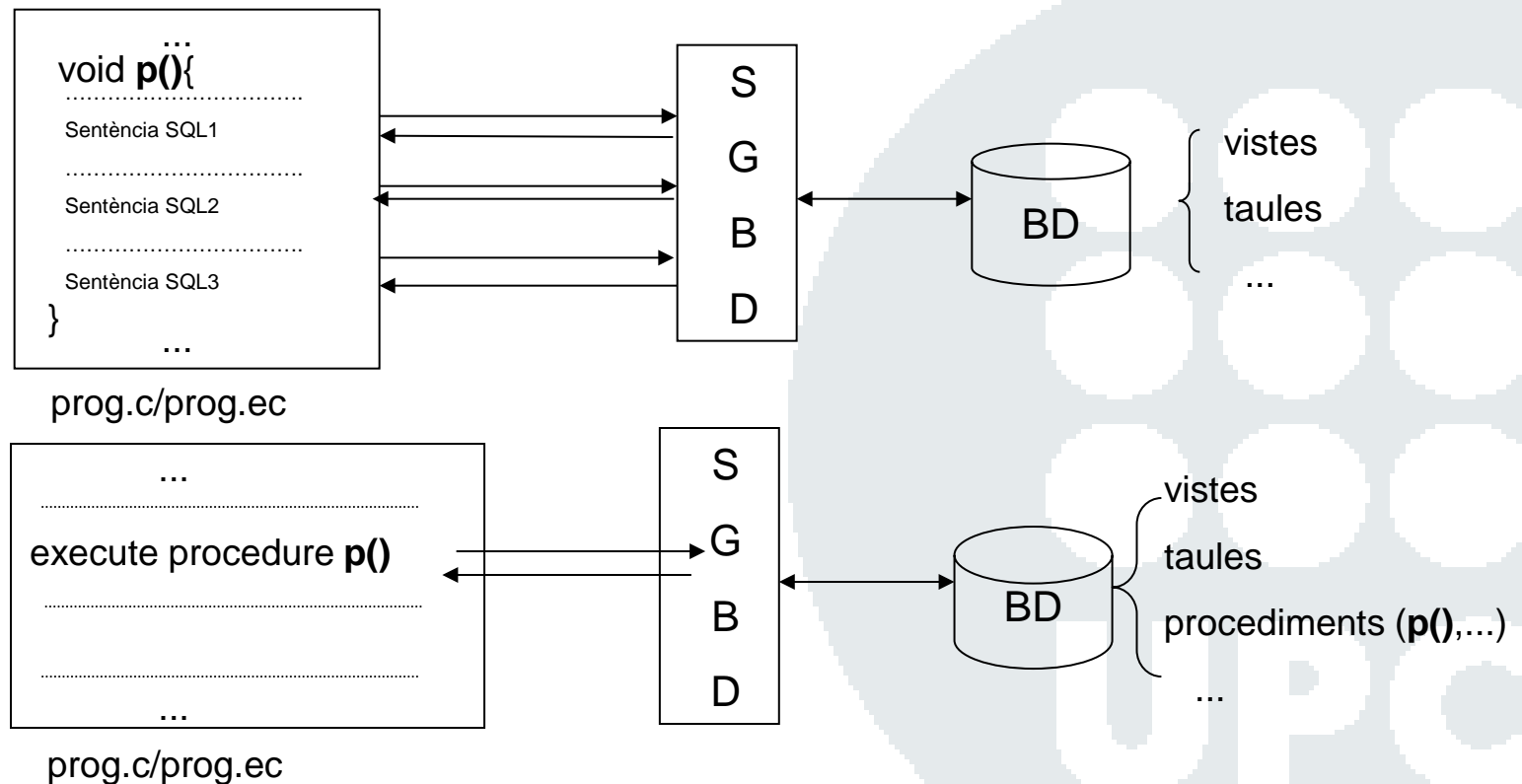
## Procediments Emmagatzemats

- Un procediment emmagatzemat és una funció definida per un usuari que, un cop creat, s'emmagatzema a la BD i es tracta com un objecte més de la BD.
- Per poder escriure procediments emmagatzemats disposem de dos tipus de sentències:
  - Sentències pròpies de l'SQL
  - Sentències pròpies del llenguatge de programació que cada SGBD té per implementar els procediments emmagatzemats (e.x: PostgreSQL de PostgreSQL)
- Els procediments emmagatzemats es poden executar:
  - De manera interactiva, amb SQL directe o interactiu
  - Des d'una aplicació
  - Des d'un altre procediment emmagatzemat

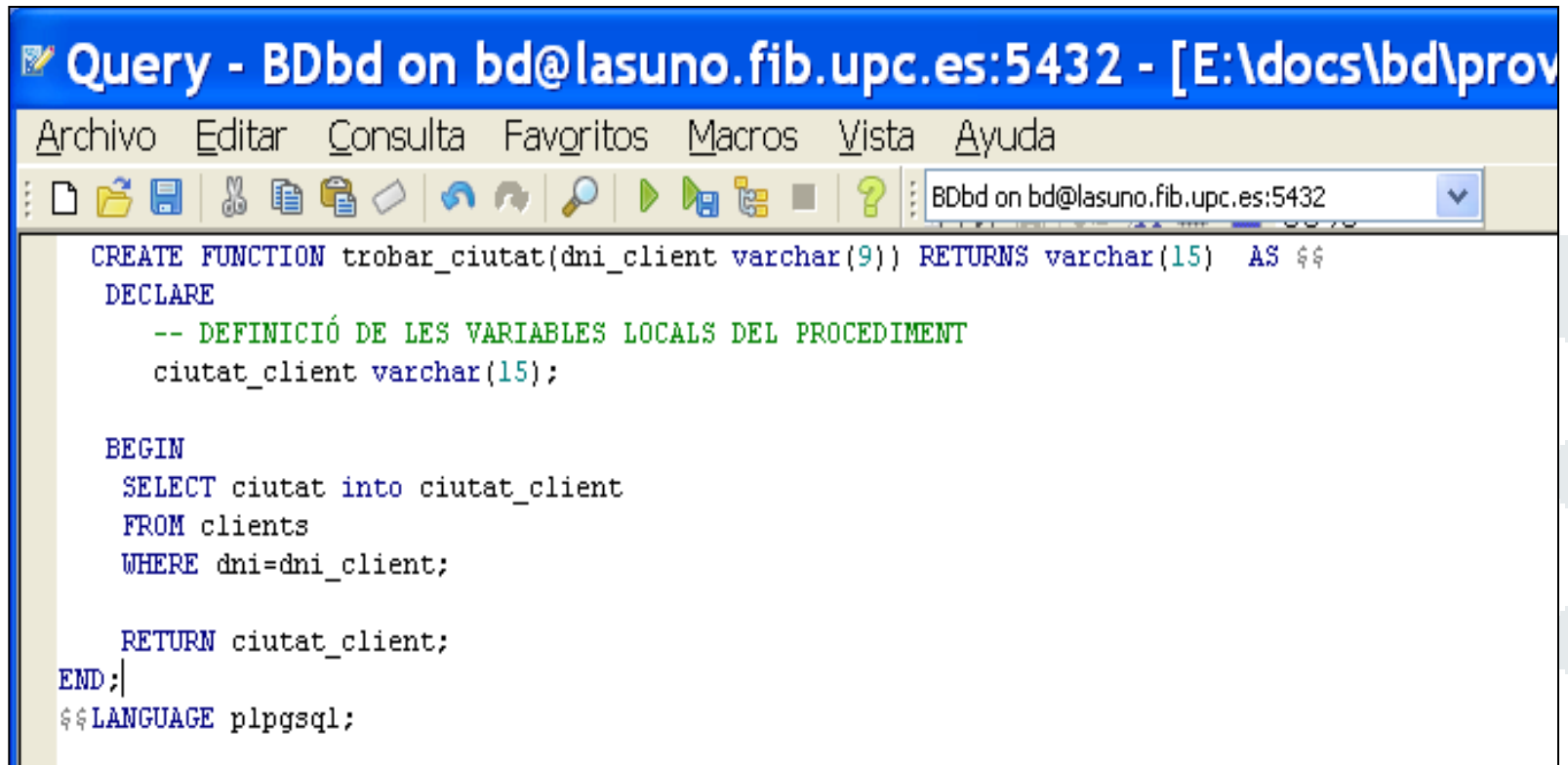


## Procediments emmagatzemats

- Serveixen per:
  - Simplificar el desenvolupament d'aplicacions
  - Millorar el rendiment de la BD
  - Controlar les operacions que els diferents usuaris realitzen contra la BD



## Procediments emmagatzemats: Exemple de creació



```
CREATE FUNCTION trobar_ciutat(dni_client varchar(9)) RETURNS varchar(15) AS $$
DECLARE
    -- DEFINICIÓ DE LES VARIABLES LOCALS DEL PROCEDIMENT
    ciutat_client varchar(15);

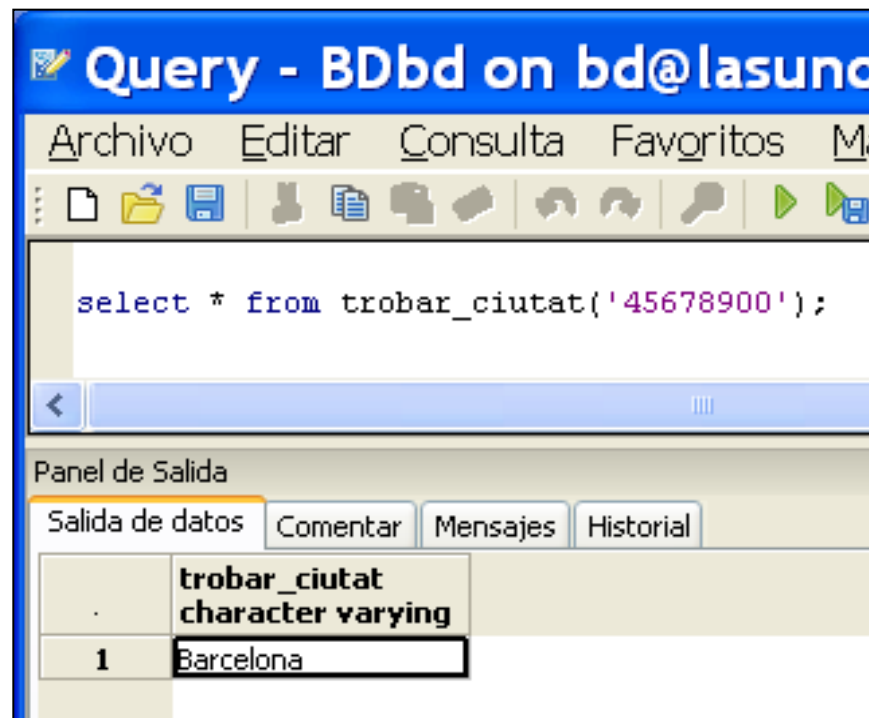
BEGIN
    SELECT ciutat into ciutat_client
    FROM clients
    WHERE dni=dni_client;

    RETURN ciutat_client;
END;
$$ LANGUAGE plpgsql;
```

Si es vol esborrar el procediment emmagatzemat cal fer:

```
DROP FUNCTION trobar_ciutat(varchar(9));
```

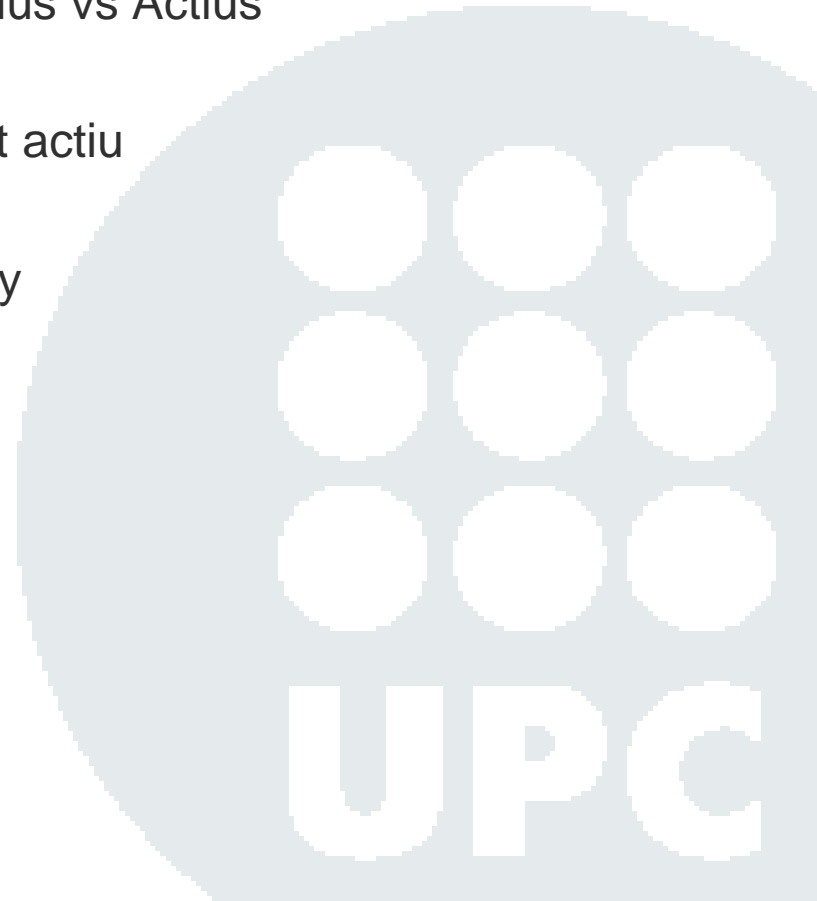
## Procediments emmagatzemats: Exemple d'execució



En aquest cas es tracta d'una execució de manera interactiva. Però com s'ha indicat abans també es pot fer execucions des de dins d'altres procediments o des d'aplicacions en general.

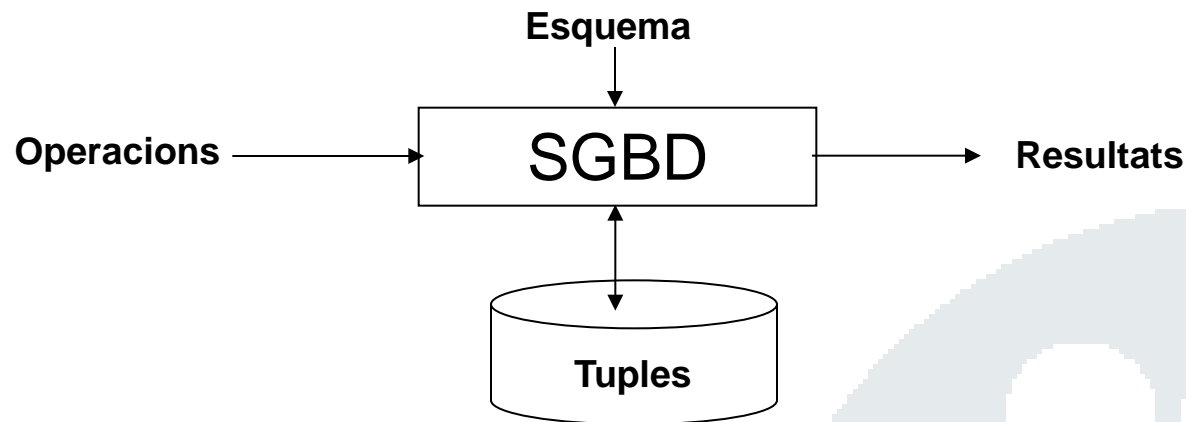
# Disparadors

- Motivació: Sistemes Passius vs Actius
- Dinàmica o comportament actiu
- Consideracions de disseny





## Disparadors: Motivació – Principis generals dels SGBD convencionals



Representació de la semàntica del món real dintre d'un marc de:

### ■ Model de dades

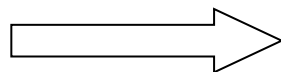
- Estructura per dades de tipus tuples
- Operacions per manipular-les
- (algunes) regles d'integritat

### ■ Model de transaccions

- Execució “sota demanda” de transaccions definides pels usuaris amb garantia de compliment d'alguns criteris (ACID)

## Disparadors: Motivació - Altres tipus de semàntiques

- Monitorització/alerta de determinades situacions → comportament reactiu
  - Auditoria d'operacions, Avisos de situacions, Regles de negoci,...
- Comprovació de restriccions d'integritat no expressables directament en el model
  - Restriccions dinàmiques, assercions,...
- Manteniment de restriccions d'integritat
- Manteniment automàtic
  - Atributs derivats, Vistes materialitzades,...
- ...

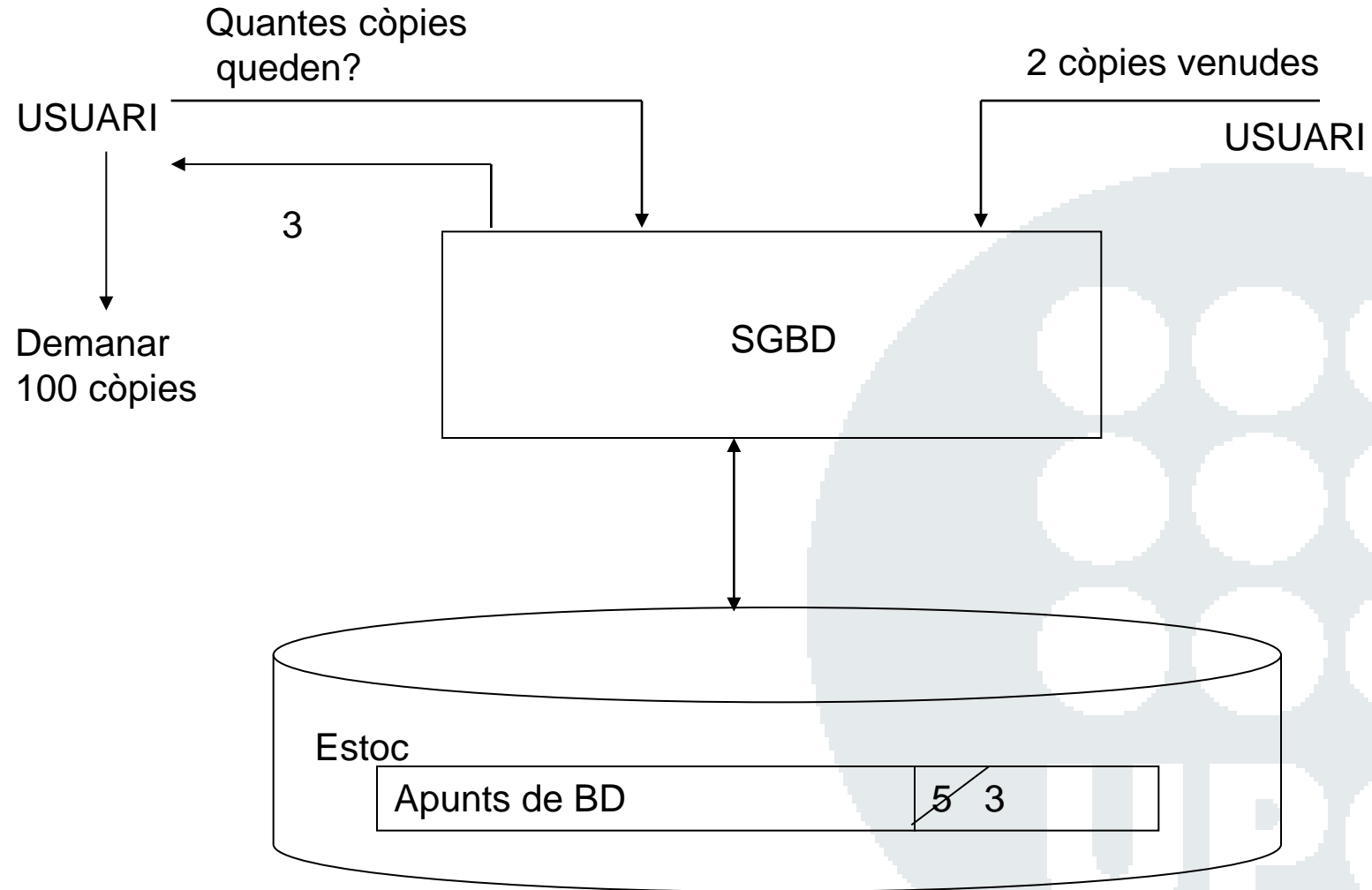


Poc suport dels SGBD convencionals !!!

## Disparadors: Motivació - Altres tipus de semàntiques: Exemples

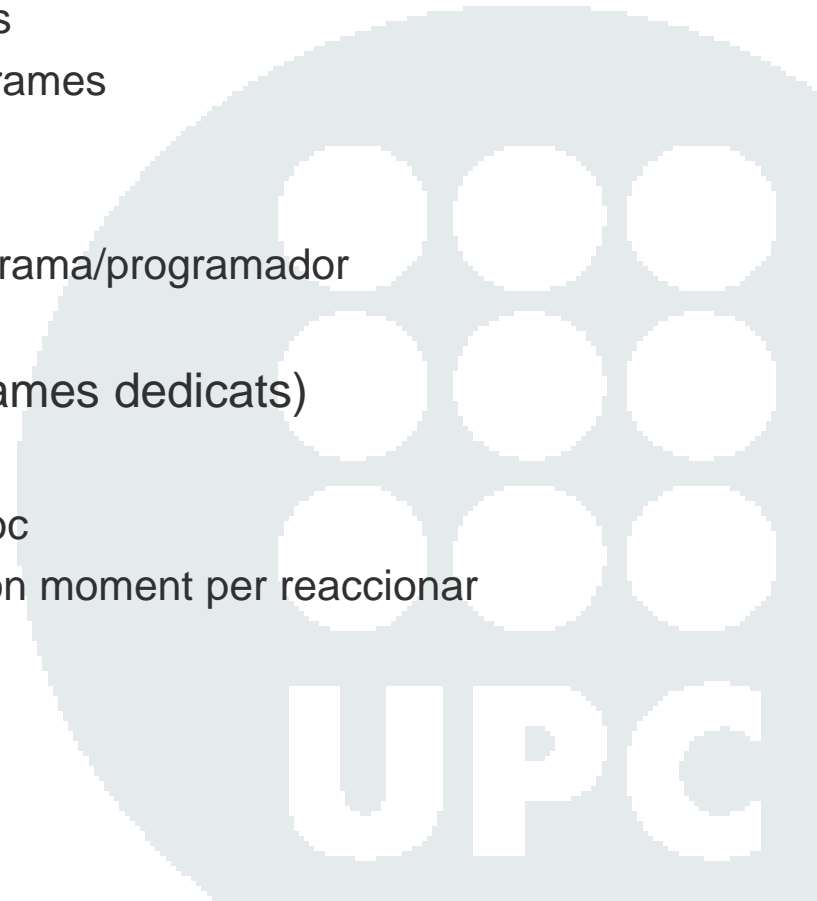
- **Auditories d'operacions.** Registrar les modificacions del sou dels empleats en una taula de la base de dades. Per cada modificació afegir una fila a la taula d'auditoria que indiqui l'usuari que ha fet la modificació, l'empleat al que se li ha modificat, l'increment/decrement, i l'instant en què s'ha fet.
- **Avisos de situacions.** Notificar als usuaris quan l'estoc d'algun producte passi a estar per sota d'un llindar.
- **Regles de negoci:** Una única sentència de modificació no pot augmentar la suma total del preu dels productes en més d'un 10%.
- **Comprovació de restriccions d'integritat no expressables directament en el model.** Tots els empleats han de treballar a un departament que està situat a la ciutat on resideixen. Quan no es compleix cal que es produeixi una excepció.
- **Manteniment de restriccions d'integritat.** No hi pot haver cap departament amb menys de 2 empleats. Quan s'esborra empleats d'un departament, deixant el amb menys de 2 empleats, per mantenir la restricció, cal esborrar també el departament.
- **Manteniment automàtic d'atributs derivats:** Atribut derivat preu de venda d'un producte que es calcula com 110% del seu preu de cost. Quan hi ha un canvi en el preu de cost cal incrementar el preu de venda tenint en compte la relació.
- **Manteniment automàtic de vistes materialitzades:** Taula que conté els empleats assignats al departament de personal. Quan s'afegeixen o esborren empleats, si són del departament de personal, cal afegir-los o esborrar-los de la taula.

## Disparadors: Motivació – Els SGBD convencionals són “Passius”

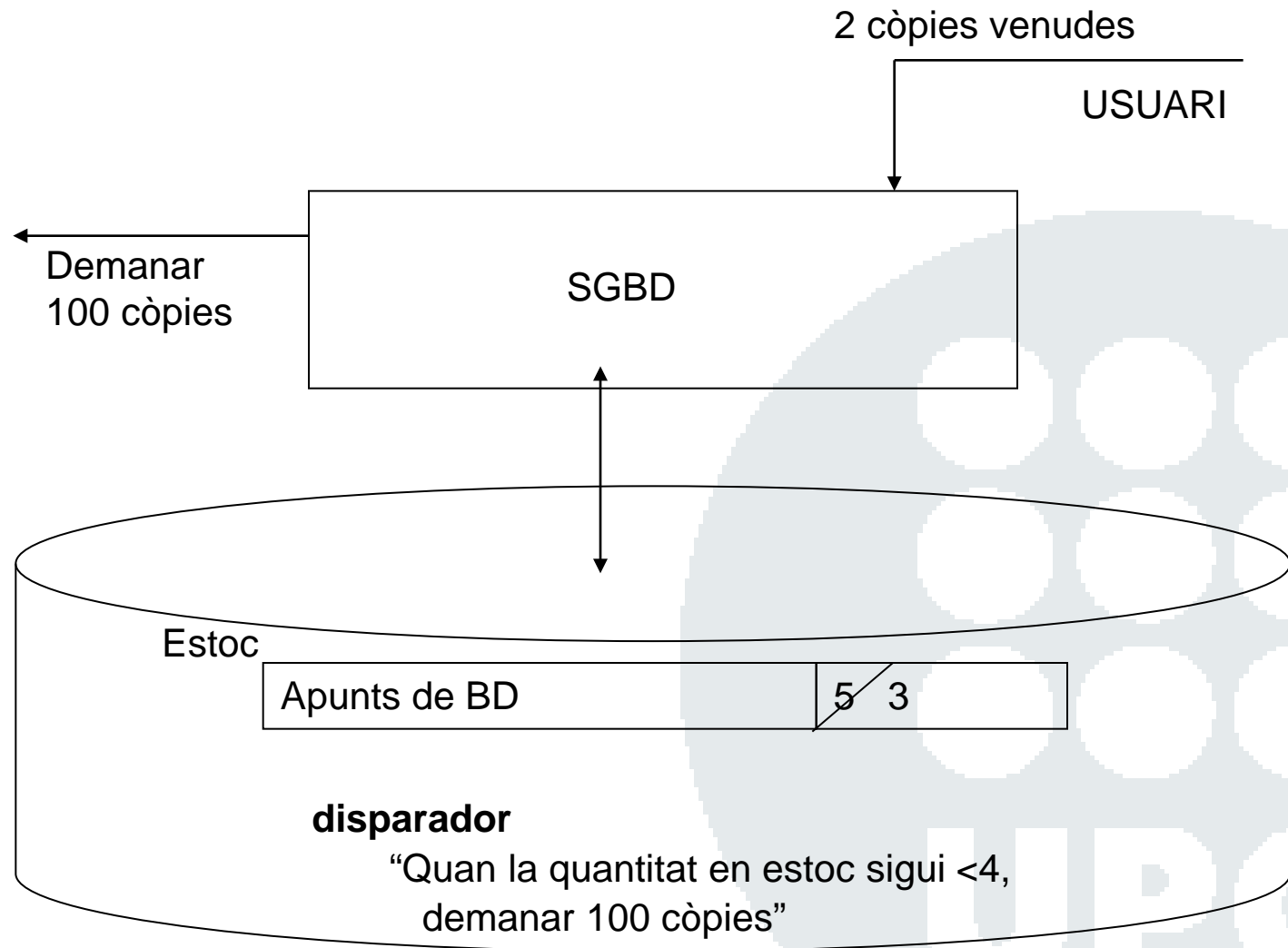


## Disparadors: Motivació – Els SGBD convencionals són “Passius”

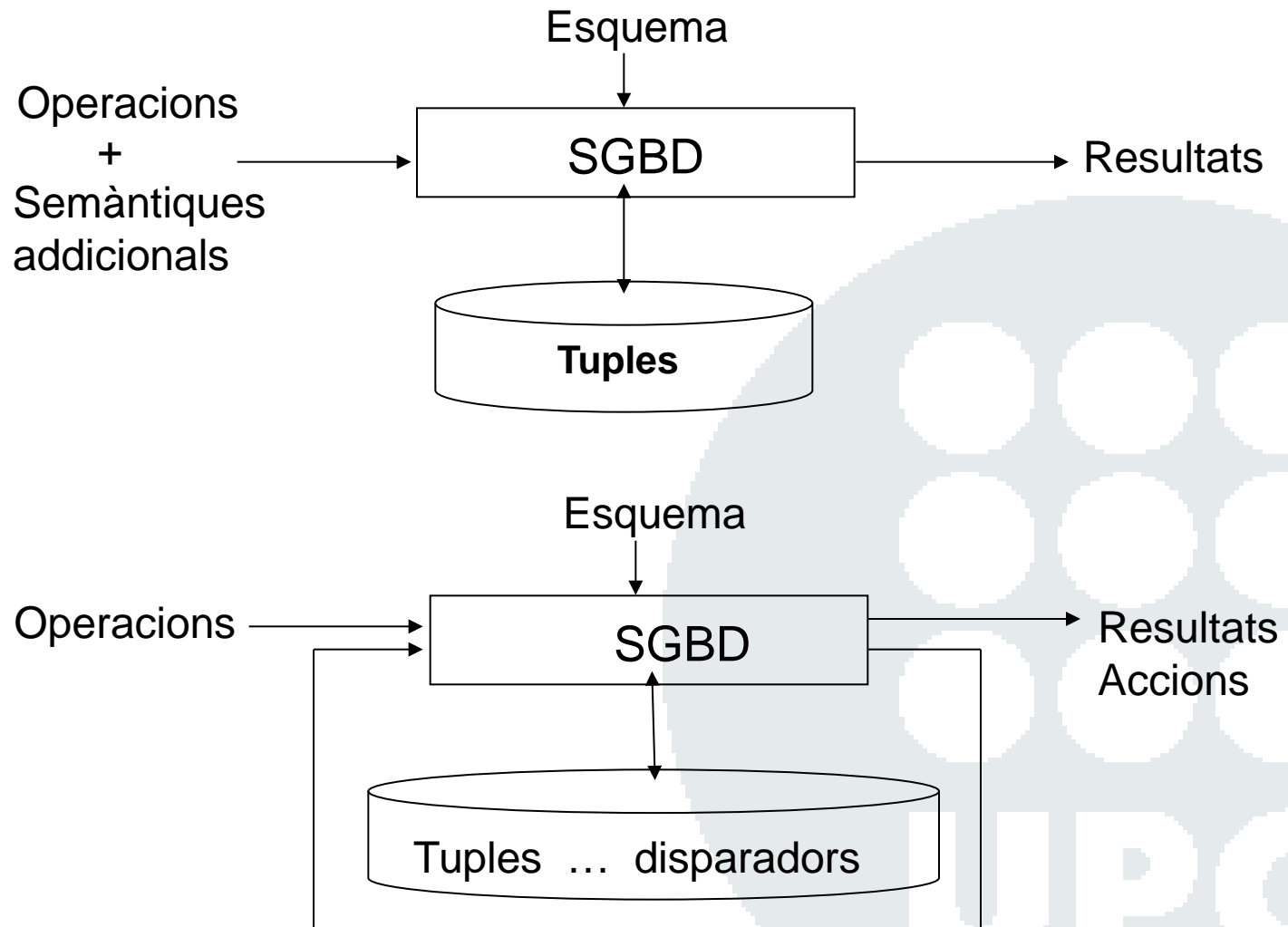
- Els programes/transaccions es preocupen de les semàntiques addicionals
  - La semàntica està:
    - Amagada en el codi dels programes
    - Distribuïda/replicada en molts programes
    - (Difícil de trobar, canviar, ...)
  - Eficàcia/correctesa depèn de cada programa/programador
- “Polling” periòdic de la BD (usant programes dedicats)
  - La semàntica està concentrada en un lloc
  - “Polling” ocasionals: podem perdre el bon moment per reaccionar
  - “Polling” freqüent: pèrdua d'eficiència



## Disparadors: Motivació – Comportament “Actiu”



## Disparadors: Motivació – SGBD “Passius” versus “Actius”



## Disparadors: Dinàmica

Un disparador és una regla ECA:

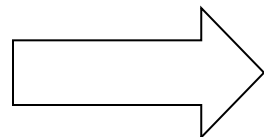
Esdeveniment	E
Condició	C
Acció	A

“Quan es produeix E, si C, aleshores executar A”

La majoria dels sistemes comercials disposen de disparadors:

Oracle, PostgreSQL, SQL Server

I a SQL:1999 es defineixen com a estàndard.



**petites diferències entre els sistemes !!!**



## Disparadors: Dinàmica

Un cop creat un trigger, el SGBD invoca automàticament l'acció indicada en resposta als esdeveniments generats per sentències executades sobre la base de dades tals com **INSERT**, **DELETE** o **UPDATE**, en cas que es compleixi la condició especificada.

Els triggers poden ser **before** o **after**:

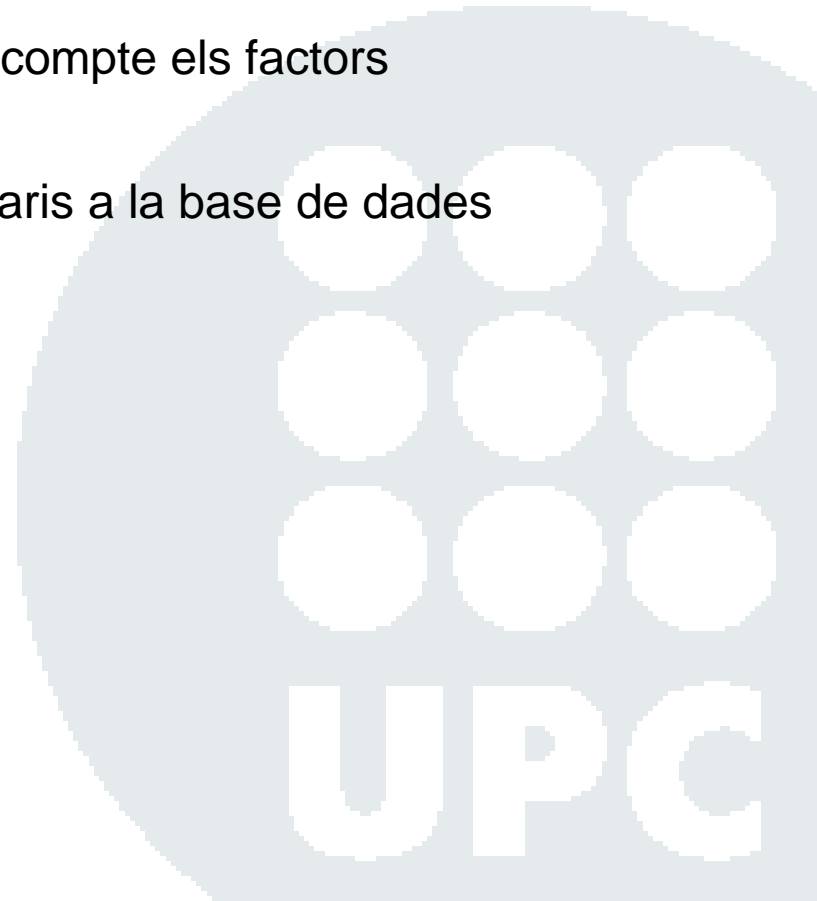
- **Before**: L'acció és invocada pel SGBD abans de l'execució de la sentència.
- **After**: L'acció és invocada pel SGBD després de l'execució de la sentència.

Els triggers poden ser **for each row** o **for each statement**:

- **For each row**: L'acció s'executa 1 vegada per cada tupla afectada per la sentència. L'execució pot dependre de les dades de la tupla afectada.
- **For each statement**: L'acció s'executa 1 vegada, independentment de quines i quantes tuples afecta la sentència.

## Disparadors: Consideracions de disseny

- Sovint existeixen diverses solucions vàlides per resoldre un mateix problema.
- En general el millor disseny té en compte els factors següents:
  - Estalviar accessos innecessaris a la base de dades
  - Evitar fer feina innecessària



## Disparadors: Disseny

- Monitorització/alerta de determinades situacions (Auditoria d'operacions, Avisos de situacions, Regles de negoci, ...)
- Comprovació de restriccions d'integritat no expressables directament en el model (Restriccions dinàmiques, Assercions, ...)
- Manteniment de restriccions d'integritat
- Manteniment automàtic d'atributs derivats, de vistes materialitzades.

- Per dissenyar disparadors, ens ajudarà tenir en compte:
  - les consideracions de disseny anteriors,
  - el tipus de semàntica a implementar,
  - el comportament que ha de tenir el disparador a implementar
- Pensar en el disseny dels disparadors corresponents als exemples de la transparència **Motivació - Altres tipus de semàntiques: Exemples**

## Disparadors: Disseny - Exemple

Comprovar la restricció “*El sou d'un empleat no pot baixar*” mitjançant disparadors.

```
CREATE TABLE empleats (  
    num_empl INTEGER PRIMARY KEY,  
    sou INTEGER NOT NULL CHECK (sou>=0 and sou<2000),  
    ....);
```

En principi, el comprovar les restriccions el més aviat possible, seria el més eficient. Això inclou les restriccions implementades amb disparadors, ja que evitaria arribar a l'executar l'esdeveniment que llança el disparador en els casos de violació de la restricció

En aquest cas => TRIGGER BEFORE / FOR EACH ROW

- Però si la sentència que dispara el disparador viola la restricció d'integritat definida en crear les taules de la base de dades CHECK (sou>=0 and sou<2000):

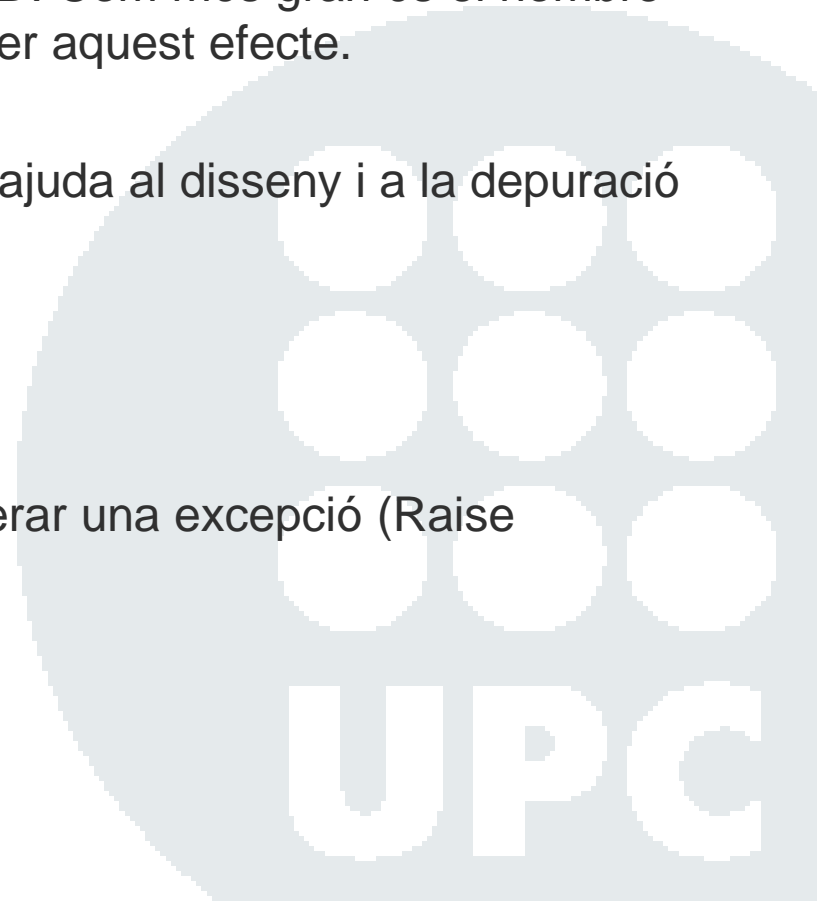
```
UPDATE empleats  
SET sou=3000  
WHERE num_empl=10;
```

- Quan seria més eficient comprovar les restriccions d'integritat de la BD? Abans o després d'executar les accions del disparador?
- Si la restricció CHECK (sou>=0 and sou<2000) es viola molt sovint, potser => TRIGGER AFTER / FOR EACH ROW

**MOLTS ASPECTES A TENIR EN COMPTE!**

## Disparadors: Consideracions de disseny

- L'efecte d'una sentència d'actualització amb la presència de disparadors depèn de l'estat de la BD. Com més gran és el nombre de disparadors, més difícil es fa saber aquest efecte.
- Seria convenient disposar d'eines d'ajuda al disseny i a la depuració dels programes, però...
- Conclusió:
  - Si l'acció del disparador és generar una excepció (Raise Exception), cap problema;
  - Altrament alerta !!!



## Disparadors: Consideracions de disseny – Cascada

Create Trigger del_a AFTER delete on a FOR EACH ROW (Execute Procedure p1())	Create Function p1() Begin <b>Delete From b....</b> End
Create Trigger del_b AFTER delete on b FOR EACH ROW (Execute Procedure p2())	Create Function p2() Begin <b>Delete From C....</b> End
Create Trigger del_c AFTER delete on c FOR EACH ROW (Execute Procedure p3())	<b>I SI?</b> Create Function p3() Begin <b>Delete From a...</b> End