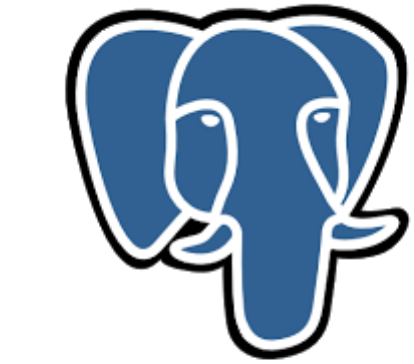
M02 - UF4

BD Objecte - Relacional



PostgreSQL

Manuel Ruiz Morante
Curs 2019-20

1. Tipus de gestors de BD

1. Relacionals
2. Orientades a objecte
3. Objecte - relacionals
4. Orientades a documents - XML, JSON, etc

2. BD Objecte relacional

Les característiques principals de les BD Objecte - Relacional són:

- Contenen les dues tecnologies, la relacional i la d'objectes
- Els conceptes en els que es basen són els mateixos que en la programació OO, és a dir: encapsulació, herència, polimorfisme, sobrecàrrega, etc.
- Poden emmagatzemar objectes que estiguin formats per altres objectes.
- Podem importar les aplicacions d'una BD Relacional sense haver de reescriure-les.

3. Com funcionen generalment?

Com ja sabem, una classe és un tipus de dades que encapsula les dades necessàries (atributs) i les funcions per accedir-hi (mètodes). Tots els objectes d'una classe tenen una còpia d'aquestes dades i funcions, és a dir, tenen **identitat pròpia**.

A una BDOR, una **classe** es representa com la **definició d'un type i/o taula** on:

- Els atributs de la classe son els camps de la taula.
- Els mètodes poden codificar-se com funcions utilitzant llenguatge de BBDD, conegut de forma general com PL (Procedural Language). PostgreSQL admet diferents llenguatges, fins i tot JAVA.

  - S'aconsella l'ús de PL quan la funcionalitat a desenvolupar requereix del tractament de molta informació, ja que el procés s'executarà directament al servidor on està el SGBD, i no s'hauràn de fer múltiples peticions des del client al servidor.

Així mateix, cada **objecte** equival a una fila/registre/tupla d'aquesta taula i existeix el camp conegut com l'identificador d'objecte (*object identifier,* **OID**), que és únic per a cada objecte.

Tal i com fem amb les PK (Primary keys) de les BD relacionals, podem utilitzar el camp **OID** per crear FK(Foreign keys) des d'altres taules.

4. Treballar amb les BBDD

4.1. Creació d'elements (DDL-Data Definition Language)

4.1.1. Taules (TABLE)

Cadascuna de les classes que hem de guardar a la BD les crearem com a taules:

```
CREATE TABLE persones (
        idpersona int,
        nif text,
        nom text,
        adreça Adreça,
        email text,
        telefon text
        ...
) WITH OIDS;
```

Si volem crear el camps **oid i tableoid** a la taula haurem d'indicar-ho amb "WITH OIDS". Aquest camps s'utilitzen com a PK (*primary keys*) a vàries taules del sistema, però també els podem crear a les nostres taules per fer-los servir com identificadors únics dels nostres objectes. No obstant, no es recomana el seu ús com a PK a les nostres taules (tot i que a la versió 12 ja no ho especifiquen).

Si hem de referenciar el camp oid d'una taula des de una altra taula, podem utilitzar el tipus de dada oid, per exemple.

```
CREATE TABLE familiar (
        oidpersona oid REFERENCES persones,
        idfamiliar int,
        nom text,
        parentesc text,
        …
);
```

Tot i que es pugui fer servir, per compatibilitat amb altres SGBD no es recomanable fer-ho.

4.1.1.1. Tipus de dades

Table 8.1. Data Types

Name	Aliases	Description
`bigint`	`int8`	signed eight-byte integer

`bigserial`	`serial8`	autoincrementing eight-byte integer
`bit [ (`***n***`) ]`		fixed-length bit string
`bit varying [ (`***n***`) ]`	`varbit [ (`***n***`) ]`	variable-length bit string
`boolean`	`bool`	logical Boolean (true/false)
`box`		rectangular box on a plane
`bytea`		binary data (“byte array”)
`character [ (`***n***`) ]`	`char [ (`***n***`) ]`	fixed-length character string
`character varying [ (`***n***`) ]`	`varchar [ (`***n***`) ]`	variable-length character string
`cidr`		IPv4 or IPv6 network address
`circle`		circle on a plane
`date`		calendar date (year, month, day)
`double precision`	`float8`	double precision floating-point number (8 bytes)
`inet`		IPv4 or IPv6 host address
`integer`	`int`, `int4`	signed four-byte integer
`interval [ `***fields***` ] [ (`***p***`) ]`		time span
`json`		textual JSON data
`jsonb`		binary JSON data, decomposed
`line`		infinite line on a plane
`lseg`		line segment on a plane
`macaddr`		MAC (Media Access Control) address

`macaddr8`		MAC (Media Access Control) address (EUI-64 format)
`money`		currency amount
`numeric [ (p, s) ]`	`decimal [ (p, s) ]`	exact numeric of selectable precision
`path`		geometric path on a plane
`pg_lsn`		PostgreSQL Log Sequence Number
`point`		geometric point on a plane
`polygon`		closed geometric path on a plane
`real`	`float4`	single precision floating-point number (4 bytes)
`smallint`	`int2`	signed two-byte integer
`smallserial`	`serial2`	autoincrementing two-byte integer
`serial`	`serial4`	autoincrementing four-byte integer
`text`		variable-length character string
`time [ (p) ] [ without time zone ]`		time of day (no time zone)
`time [ (p) ] with time zone`	`timetz`	time of day, including time zone
`timestamp [ (p) ] [ without time zone ]`		date and time (no time zone)
`timestamp [ (p) ] with time zone`	`timestamptz`	date and time, including time zone
`tsquery`		text search query
`tsvector`		text search document

`txid_snapshot`		user-level transaction ID snapshot
`uuid`		universally unique identifier
`xml`		XML data

4.1.1.2. PK CONSTRAINTS (Primary keys)

Recordeu que, per garantir la integritat de les dades, hem de definir tant les PK com les FK.
Definim la **clau primària** per assegurar que els registre de la nostra taula s'identifiquen de forma única respecte als altres. La clau primària pot estar formada per una única columna o per més d'una, **que és el més habitual, exemple**:

```
CREATE TABLE persones (
        idpersona int,
        PRIMARY KEY(idpersona),
        …
);
```

També:

```
CREATE TABLE persones (
        idpersona int PRIMARY KEY,
        …
);
```

Taula amb PK múltiple:

```
CREATE TABLE familiars (
        idpersona int,
        idfamiliar int,
        nom text,
        parentesc text,
        PRIMARY KEY(idpersona, idfamiliar),
        …
);
```

4.1.1.3. FK CONSTRAINTS (Foreign keys)

Definim una **clau forània** cap a una altra taula per assegurar que el valor d'una columna és correcte, és a dir, existeix a la taula referenciada. Hi ha 3 opcions de fer-ho:

```
CREATE TABLE familiars (
        idpersona int REFERENCES persones,
```

```
        idfamiliar int,
        nom text,
        parentesc text,
        PRIMARY KEY(idpersona, idfamiliar),
        ...
);

CREATE TABLE familiars (
        idpersona int REFERENCES persones(idpersona),
        idfamiliar int,
        nom text,
        parentesc text,
        PRIMARY KEY(idpersona, idfamiliar),
        ...
);

CREATE TABLE familiars (
        idpersona int,
        idfamiliar int,
        nom text,
        parentesc text,
        PRIMARY KEY(idpersona, idfamiliar),
        FOREIGN KEY idpersona REFERENCES persones(idpersona)
...
);
```

A més, podem realitzar unes determinades accions sobre el registres que fan la referència en cas de ELIMINACIÓ (**ON DELETE**) o ACTUALITZACIÓ (**ON UPDATE**) del valor referenciat:

- RESTRICT: no es permet eliminar/actualitzar el registre de la taula origen si existeixen registres que fan referència
- NO ACTION (per defecte): igual que RESTRICTION però s'avalua al final de la transacció. La majoria de SGBD no fan cap diferència entre una i una altra.
- CASCADE: elimina/actualitza els registres de la taula amb la FK si s'elimina/actualitza el registre a la taula origen
- SET NULL o SET DEFAULT: el camp es posaria a NULL o al seu valor per defecte en cas que s'eliminés el registre origen.

```
CREATE TABLE familiars (
        idpersona int REFERENCES persones ON DELETE CASCADE,
        idfamiliar int,
        nom text,
        parentesc text,
        PRIMARY KEY(idpersona, idfamiliar),
);
```

[4.1.1.4. ALTRES CONSTRAINTS]

4.1.2. Tipus d'objectes (TYPE)

El primer cas en el que crearem un nou tipus d'objecte serà sempre que tinguem una propietat d'una classe d'un tipus d'una altre classe creada per nosaltres, per exemple la propietat **adreça** de la classe Persona (codi JAVA):

```
public class Persona {
        private String idpersona;
        private String nif;
        private String nom;
        private Adreça adreça;
        private String email;
        private String telefon;
        ...
}

public class Adreça {
        private String poblacio;
        private String provincia;
        private String cp;
        private String domicili;
        …
}
```

Com que per crear la taula **persones** hem de crear un camp del tipus **Adreça**. hem de crear-lo abans amb:

```
CREATE TYPE adreça AS (
        poblacio text,
        provincia text,
        cp text,
        domicili text
);
```

I després ja podem fer:

```
CREATE TABLE persones (
        idpersona int PRIMARY KEY,
        nif text,
        nom text,
        adreça adreça,
        …
);
```

4.1.3. Crear taules a partir de tipus d'objectes

El segon cas en el que serà útil crear un tipus d'objecte serà quan, segons el nostres requeriments, haguem de definir una taula amb els mateixos camps vàries vegades. Per exemple en casos en que la informació varia d'any en any o temporada a temporada, podríem crear diferents taules per cada any de la següent manera:

Donat el tipus d'objecte professor:

```
CREATE TYPE professor AS (
        idprofessor int,
        nom text,
        materia text,
);
```

Podríem crear taules diferents per cada curs a partir del tipus definit:

```
CREATE TABLE professors_1617 OF professor (
        PRIMARY KEY (idprofessor)
);

CREATE TABLE professors_1718 OF professor (
        PRIMARY KEY (idprofessor)
);
```

4.1.4. Herència

Donada una relació d'herència entre dues classes, per exemple Client i Persona, podem crear la taula **clients** de la següent manera:

```
CREATE TABLE clients (
        enviarpublicitat boolean,
        PRIMARY KEY (idpersona)  → PK pot ser igual o diferent a la taula mare
) INHERITS (persones);
```

Si féssim una consulta a la taula **clients**, veuríem també les columnes de la taula **persones**. En aquest cas l'identificador d'objecte (**oid**) serà el mateix per un mateix objecte a les dues taules.

Si volem evitar que es puguin crear registres/objectes a la taula mare (comportament semblant al *abstract* de JAVA), podem crear una CONSTRAINT de tipus **CHECK** (s'avalua a inserts i updates) que retorni sempre false i que no s'hereti a les taules filles:

```
CREATE TABLE persones (
        idpersona int PRIMARY KEY,
        nif text,
        nom text,
        CHECK (false) NO INHERIT,
        …
```

```
);
```

4.1.5. Col·leccions (arrays)

Les col·leccions són un tipus de dada especial que ens permet emmagatzemar un conjunt d'elements dintre d'una columna de la taula.

```
CREATE TABLE persones (
        idpersona int PRIMARY KEY,
        nif text,
        nom text,
        adreça Adreça,
        email text,
        telefons varchar(9)[],
        ...
);

CREATE TABLE persones (
        idpersona int PRIMARY KEY,
        nif text,
        nom text,
        adreça Adreça,
        email text,
        telefons varchar(9) ARRAY, → notació estàndard SQL
        ...
);
```

Com s'ha de fer als SGBD que no suporten arrays? Doncs creant una subtaula amb un FK cap a la taula superior.

```
CREATE TABLE telefons (
        idpersona int REFERENCES persones,
        telefon varchar(9),
        PRIMARY KEY (idpersona, telefon)
);
```

4.2. Manipulació de dades (DML-Data Manipulation Language)

```
CREATE TYPE adreça AS (
        poblacio text,
        provincia text,
        cp text,
        domicili text
);
```

```
CREATE TABLE persones (
        idpersona int PRIMARY KEY,
        nif text,
        nom text,
        adreça adreça,
        email text,
        telefons varchar(9) ARRAY,
        ...
);
```

Utilitzant com a referència la definició de la taula **persones** de dalt, anem a veure com fer insercions, consultes, actualitzacions i eliminacions de registres on intervinguin els camps **adreça** i **telèfons**, així com el tractament de l'herència en cada cas.

Amb la resta de camps el funcionament és el que ja heu estudiat a les anteriors UF del mòdul.

- Recordeu que, parlem de **transacció** quan hi ha diverses sentències i es considera que s'han d'executar en bloc per mantenir la integritat de les dades. Les transaccions ens permeten tornar les dades de la BD a un estat anterior (coherent) si alguna sentència ha fallat.

4.2.1. Inserció (INSERT)

Utilitzarem ROW i ARRAY per inserir els valors d'una propietat d'un **TYPE** i una **col·lecció**, respectivament.

Exemple:

```
INSERT INTO persones (idpersona, nif, nom, adreça, email, telefons)
VALUES (1, '12345678A', 'Paco', ROW('Badia del Vallès', 'Barcelona',
'08214','C\Mallorca s/n'),'paco@ibadia.cat', ARRAY['937101010', '699889977'])
```

Si s'insereixen tots els valors, no cal especificar els camps de la taula:

```
INSERT INTO persones
VALUES (1, '12345678A', 'Paco', ROW('Badia del Vallès', 'Barcelona',
'08214','C\Mallorca s/n'),'paco@ibadia.cat', ARRAY['937101010', '699889977'])
```

Si tinguéssim la següent estructura de dades:

```
CREATE TYPE telefon AS (
        tipus text,
        numero varchar(9)
);
```

```
CREATE TABLE persones (
        idpersona int,
        nif text,
        nom text,
        adreça Adreça,
        email text,
        telefons telefon ARRAY, → array de tipus de dada telèfon
);
```

Combinaríem ARRAY i ROW de la següent manera per inserir els telèfons:

```
INSERT INTO persones
VALUES (1, '12345678A', 'Paco', (ROW('Badia del Vallès', 'Barcelona',
'08214','C\Mallorca s/n'),'paco@ibadia.cat',
ARRAY[ROW('Fixe','937101010')::telefon,ROW('Mobil','699889977')::telefon])
```

4.2.2. Consulta (SELECT)

4.2.2.1. Herència

Consulta	Resultat
SELECT * FROM persones	Selecciona totes les files de la taula pare, incloent totes les files de les taules filles.
SELECT * FROM ONLY persones	Selecciona només les files de la taula mare.
OPCIÓ 1: SELECT pg_class.relname, persones.* FROM persones, pg_class WHERE persones.tableoid = pg_class.oid; OPCIÓ 2: SELECT tableoid::regclass, * FROM persones	Mostrar una columna amb el nom de la **taula filla** a la que pertany cada fila.
SELECT persones.tableoid::regclass, * FROM persones NATURAL FULL JOIN clients, proveidors	Mostrar una columna amb el nom de la **taula filla** a la que pertany cada fila **+** totes les columnes de les taules filles. Es posarà **NULL** a les columnes no trobades a la taula filla a la que pertany la fila.
...	...

4.2.2.2. Tipus d'objectes

Per fer referència a una columna d'un tipus d'objecte (TYPE) a qualsevol part d'una consulta, hem d'utilitzar el parèntesis per indicar que es tracta d'un TYPE i no del nom d'una taula:

SELECT adreça.poblacio FROM persones

SELECT * FROM persones WHERE adreça.poblacio = 'Badia del Vallès'

4.2.2.3. Col·leccions (ARRAY)

Operació	Consulta
Tots els telèfons	SELECT telefons FROM persones
Accedir al primer telèfon del ARRAY	SELECT telefons[1] FROM persones
Cada telèfon com una fila	SELECT unnest(telefons) FROM persones
Longitud del array de telèfons. 1 indica la dimensió que volem consultar, en aquest cas només tenim una.	SELECT array_length(telefons, 1) FROM persones
Buscar persones que coincideixin amb un telèfon concret	SELECT * FROM persones WHERE 937101010 = ANY(telefons)
...	...

Més funcions

4.2.3. Modificació (UPDATE)

4.2.3.1. Tipus d'objectes

UPDATE persones SET adreça = ROW('Badia del Vallès', 'Barcelona', '08214','C\Mallorca s/n') WHERE ...

UPDATE persones SET adreça.poblacio = 'Badia del Vallès' WHERE ...

4.2.3.2. Col·leccions

UPDATE persones SET telefons = ARRAY['937101010', '699889977'] WHERE ...

UPDATE persones SET telefons[1] = '937101010' WHERE ...

En el cas d'un ARRAY de TYPEs
UPDATE persones SET telefons[1].numero = '937101010' WHERE ...

4.2.4. Eliminacions (DELETE)

En el cas d'eliminacions no canvia res respecte al que ja heu après anteriorment a la resta de UF's.

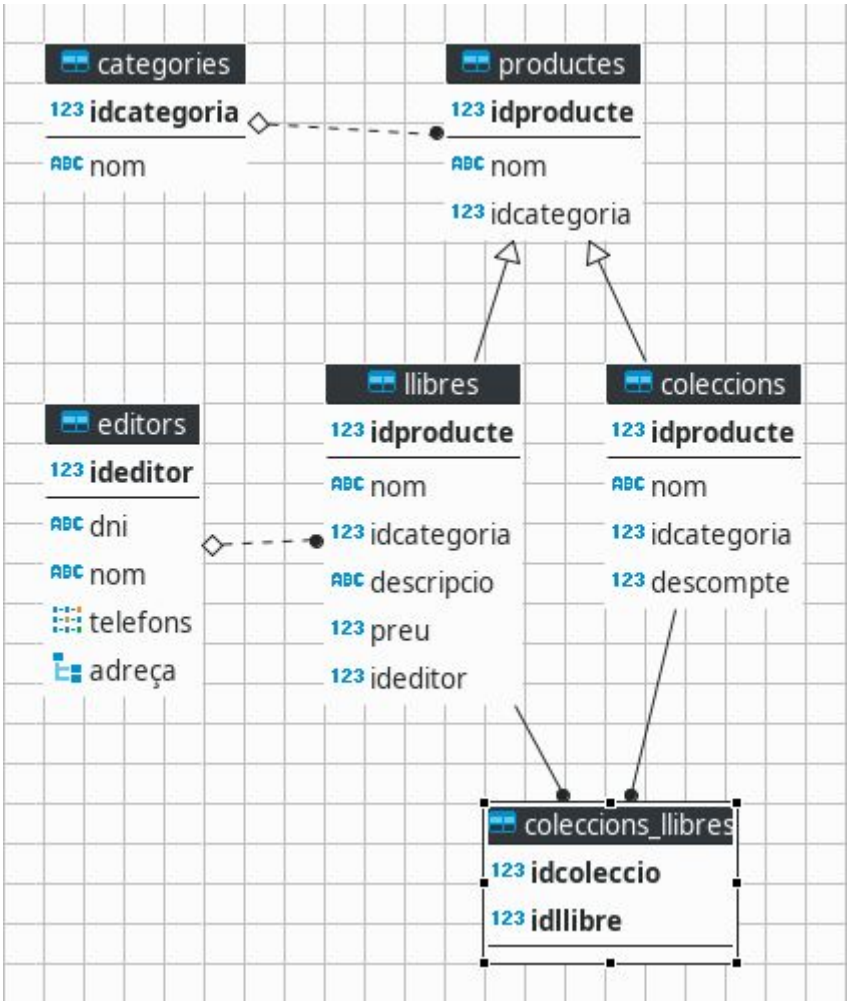
```
DELETE FROM persones WHERE ….
```

4.3. Bones pràctiques

 Conjunt de recomanacions a l'hora de nombrar les taules i columnes d'una BD que us serviran per ser més eficaços i eficients, a la vegada que facilitareu el treball a altres programadors:

- El nom de les taules ha d'estar preferentment en **plural** ja que emmagatzemen *n* objectes/files d'un tipus. De totes maneres, no es tracta d'una norma escrita, així que podeu fer servir també el singular. El que sí és obvi que heu de fer servir sempre la regla que es faci servir a l'empresa on treballeu.
- El nom de les propietats que formen les **PK** s'han de nomenar incloent "id" o un altre text identificatiu, ja sigui al davant o al darrere:
  idpersona
  personaid
  persona_id
- El nom de les propietats d'una **FK** han de ser els mateixos que el de la corresponent propietat de la taula a la que fan referència.
  persones.idpersona ←→ familiars.idpersona

- Utilitzar JOINs per accedir a les dades d'una altra taula i no subconsultes, que s'han d'utilitzar per obtenir un subconjunt d'elements. Exemple, donada la següent BD d'una llibreria:



Calcula el precio total (con descuento) del libro llamado "uno" perteneciente a la colección con identificacion "4"

AMB SUBCONSULTES

```
SELECT preu - (preu)*(select descompte from coleccions where idproducte=(select
idcoleccio from coleccions_llibres where idllibre=(select idproducte from llibres where
nom='uno') and idcoleccio=4)) as precio_en_coleccion
FROM llibres
WHERE nom='uno'
```

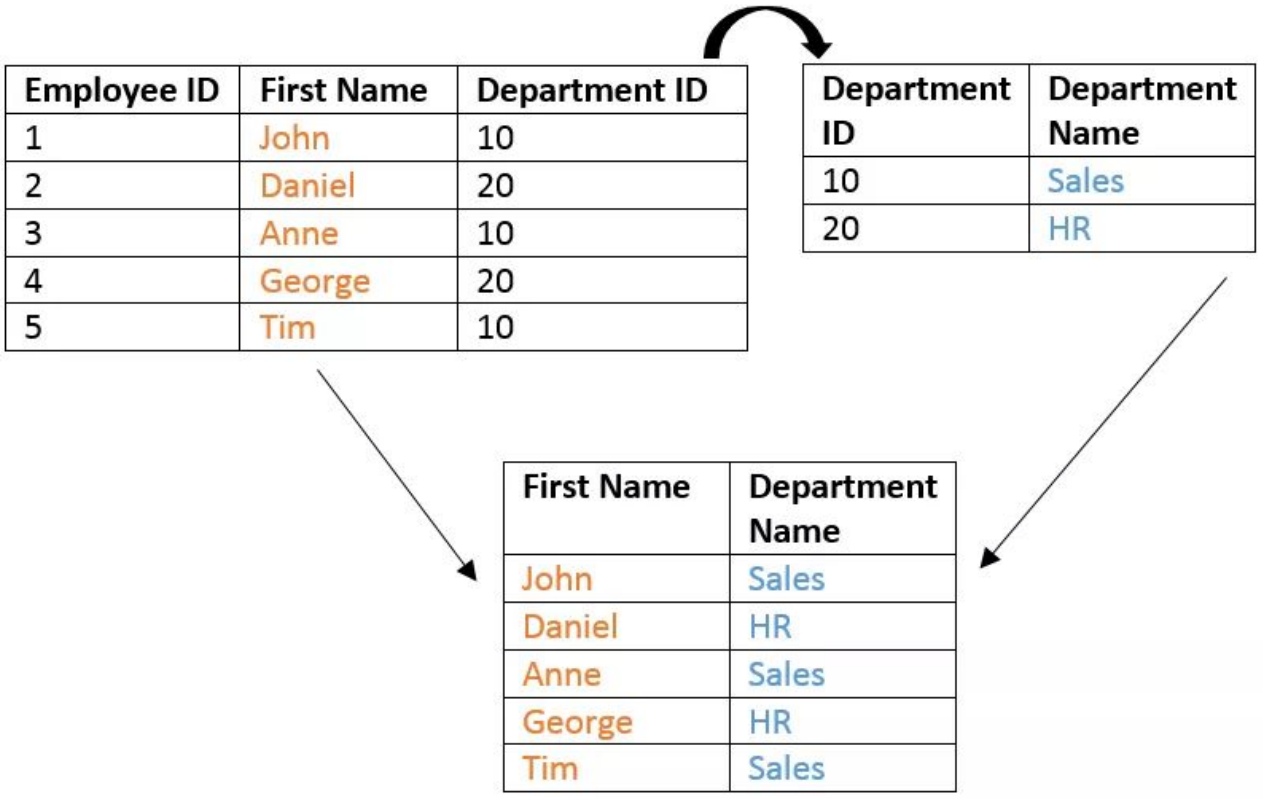
AMB JOIN

```
SELECT (preu - (preu*descompte)) as preu_collecio, preu, descompte
FROM llibres, coleccions_llibres, coleccions
WHERE llibres.nom='uno' and coleccions_llibres.idcoleccio = 4
and coleccions_llibres.idllibre=llibres.idproducte
and coleccions_llibres.idcoleccio = coleccions.idproducte
```

- Utilitzar quan sigui possible JOIN o EXIST en comptes de IN, ja que són més eficients.

- Utilitzar JOIN (amb ON, USING o *table list)* si necessitem fer coincidir les files de dues taules o un SEMI-JOIN (amb EXISTS o IN) si només necessitem la informació d'una taula perquè és més eficient.

  JOIN



Employee ID	First Name	Department ID
1	John	10
2	Daniel	20
3	Anne	10
4	George	20
5	Tim	10

Department ID	Department Name
10	Sales
20	HR

First Name	Department Name
John	Sales
Daniel	HR
Anne	Sales
George	HR
Tim	Sales

  SEMI-JOIN

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Production

Dept

DeptName	Manager
Sales	Bob
Sales	Thomas
Production	Katie
Production	Mark

Employee ⋉ Dept

Name	EmpId	DeptName
Sally	2241	Sales
Harriet	2202	Production

5. Webgrafia

- *PostgreSQL- The SQL Language* https://www.postgresql.org/docs/9.6/static/sql.html
- *PostgreSQL*

https://www.postgresql.org/
- *PostgreSQL-es*
  http://www.postgresql.org.es/
- *Wikipedia - Mapeo objeto relacional*
  https://es.wikipedia.org/wiki/Mapeo_objeto-relacional
- *Using or not using the OID in PostgreSQL*
  http://philmcrew.com/oid.html
- *IOC - Bases de dades objecte-relacionals*
  https://ioc.xtec.cat/materials/FP/Materials/IC_S_INF/INF_IC_S_M02/web/html/WebContent/u8/resum.html
- *Cómo nombrar las tablas en mi base de datos*
  http://letrascarlos.blogspot.com.es/2012/11/como-nombrar-las-tablas-en-mi-base-de.html
- *Nombre de tablas*
  http://librosweb.es/foro/pregunta/84/nombre-de-tablas/
- *Comparison of **relational** database management systems*
  https://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems
- *Comparison of **object** database management systems*
  https://en.wikipedia.org/wiki/Comparison_of_object_database_management_systems
- *Comparison of **object-relational** database management systems*
  https://en.wikipedia.org/wiki/Comparison_of_object-relational_database_management_systems